Free Component Library (FCL): Reference guide.

Reference guide for FCL units.
Document version 3.2.4
October 2026

Michaël Van Canneyt

Contents

0.1 Overview . . . 154

1 Reference for unit 'AdvancedIPC' 155

1.1 Used units . . . 155
1.2 Overview . . . 155
1.3 Constants, types and variables . . . 155
1.3.1 Resource strings . . . 155
1.3.2 Constants . . . 156
1.3.3 Types . . . 156
1.4 TMessageHeader . . . 156
1.5 EICPException . . . 156
1.5.1 Description . . . 156
1.6 TIPCBase . . . 157
1.6.1 Description . . . 157
1.6.2 Method overview . . . 157
1.6.3 Property overview . . . 157
1.6.4 TIPCBase.FindRunningServers . . . 157
1.6.5 TIPCBase.ServerRunning . . . 157
1.6.6 TIPCBase.ServerID . . . 158
1.6.7 TIPCBase.Global . . . 158
1.6.8 TIPCBase.MessageVersion . . . 158
1.7 TIPCClient . . . 158
1.7.1 Description . . . 158
1.7.2 Method overview . . . 159
1.7.3 TIPCClient.Create . . . 159
1.7.4 TIPCClient.PostRequest . . . 159
1.7.5 TIPCClient.SendRequest . . . 159
1.7.6 TIPCClient.PeekResponse . . . 160
1.7.7 TIPCClient.DeleteRequest . . . 160
1.7.8 TIPCClient.ServerRunning . . . 160
1.8 TIPCServer . . . 161

1.8.1 Description . . . 161
1.8.2 Method overview . . . 161
1.8.3 Property overview . . . 161
1.8.4 TIPCServer.Create . . . 161
1.8.5 TIPCServer.Destroy . . . 161
1.8.6 TIPCServer.PeekRequest . . . 162
1.8.7 TIPCServer.ReadRequest . . . 162
1.8.8 TIPCServer.DeleteRequest . . . 162
1.8.9 TIPCServer.PostResponse . . . 163
1.8.10 TIPCServer.FindHighestPendingRequestId . . . 163
1.8.11 TIPCServer.GetPendingRequestCount . . . 163
1.8.12 TIPCServer.StartServer . . . 163
1.8.13 TIPCServer.StopServer . . . 164
1.8.14 TIPCServer.DeletePendingRequests . . . 164
1.8.15 TIPCServer.Active . . . 164
1.9 TReleaseHandleStream . . . 164
1.9.1 Description . . . 164
1.9.2 Method overview . . . 164
1.9.3 TReleaseHandleStream.Destroy . . . 165

2 Reference for unit 'AdvancedSingleInstance' 166
2.1 Used units . . . 166
2.2 Overview . . . 166
2.3 Constants, types and variables . . . 166
2.3.1 Types . . . 166
2.4 TAdvancedSingleInstance . . . 167
2.4.1 Description . . . 167
2.4.2 Method overview . . . 167
2.4.3 Property overview . . . 167
2.4.4 TAdvancedSingleInstance.Create . . . 167
2.4.5 TAdvancedSingleInstance.Start . . . 168
2.4.6 TAdvancedSingleInstance.Stop . . . 168
2.4.7 TAdvancedSingleInstance.ServerCheckMessages . . . 168
2.4.8 TAdvancedSingleInstance.ClientPostParams . . . 168
2.4.9 TAdvancedSingleInstance.ClientPostCustomRequest . . . 169
2.4.10 TAdvancedSingleInstance.ClientSendCustomRequest . . . 169
2.4.11 TAdvancedSingleInstance.ServerPostCustomResponse . . . 169
2.4.12 TAdvancedSingleInstance.ClientPeekCustomResponse . . . 170
2.4.13 TAdvancedSingleInstance.ID . . . 170
2.4.14 TAdvancedSingleInstance.Global . . . 170

2.4.15 TAdvancedSingleInstance.OnServerReceivedCustomRequest 170

3 Reference for unit 'ascii85' 171

3.1 Used units 171
3.2 Overview 171
3.3 Constants, types and variables 171
3.3.1 Types 171
3.4 TASCII85DecoderStream 172
3.4.1 Description 172
3.4.2 Method overview 172
3.4.3 Property overview 172
3.4.4 TASCII85DecoderStream.Create 172
3.4.5 TASCII85DecoderStream.Decode 173
3.4.6 TASCII85DecoderStream.Close 173
3.4.7 TASCII85DecoderStream.ClosedP 173
3.4.8 TASCII85DecoderStream.Destroy 173
3.4.9 TASCII85DecoderStream.Read 174
3.4.10 TASCII85DecoderStream.Seek 174
3.4.11 TASCII85DecoderStream.BExpectBoundary 174
3.5 TASCII85EncoderStream 174
3.5.1 Description 174
3.5.2 Method overview 175
3.5.3 Property overview 175
3.5.4 TASCII85EncoderStream.Create 175
3.5.5 TASCII85EncoderStream.Destroy 175
3.5.6 TASCII85EncoderStream.Write 175
3.5.7 TASCII85EncoderStream.Width 176
3.5.8 TASCII85EncoderStream.Boundary 176
3.6 TASCII85RingBuffer 176
3.6.1 Description 176
3.6.2 Method overview 176
3.6.3 Property overview 176
3.6.4 TASCII85RingBuffer.Write 177
3.6.5 TASCII85RingBuffer.Read 177
3.6.6 TASCII85RingBuffer.FillCount 177
3.6.7 TASCII85RingBuffer.Size 177

4 Reference for unit 'AVL_Tree' 178

4.1 Used units 178
4.2 Overview 178
4.3 Constants, types and variables 178

4.3.1 Types . . . 178
4.3.2 Variables . . . 179
4.4 TAVLTree . . . 179
4.4.1 Description . . . 179
4.4.2 Method overview . . . 180
4.4.3 Property overview . . . 181
4.4.4 TAVLTree.Create . . . 181
4.4.5 TAVLTree.CreateObjectCompare . . . 181
4.4.6 TAVLTree.Destroy . . . 181
4.4.7 TAVLTree.SetNodeManager . . . 181
4.4.8 TAVLTree.NewNode . . . 182
4.4.9 TAVLTree.DisposeNode . . . 182
4.4.10 TAVLTree.Add . . . 182
4.4.11 TAVLTree.AddAscendingSequence . . . 183
4.4.12 TAVLTree.Delete . . . 183
4.4.13 TAVLTree.Remove . . . 183
4.4.14 TAVLTree.RemovePointer . . . 184
4.4.15 TAVLTree.MoveDataLeftMost . . . 184
4.4.16 TAVLTree.MoveDataRightMost . . . 184
4.4.17 TAVLTree.Clear . . . 184
4.4.18 TAVLTree.FreeAndClear . . . 185
4.4.19 TAVLTree.FreeAndDelete . . . 185
4.4.20 TAVLTree.Equals . . . 185
4.4.21 TAVLTree.IsEqual . . . 185
4.4.22 TAVLTree.Assign . . . 186
4.4.23 TAVLTree.Compare . . . 186
4.4.24 TAVLTree.Find . . . 186
4.4.25 TAVLTree.FindKey . . . 186
4.4.26 TAVLTree.FindNearestKey . . . 187
4.4.27 TAVLTree.FindSuccessor . . . 187
4.4.28 TAVLTree.FindPrecessor . . . 187
4.4.29 TAVLTree.FindLowest . . . 188
4.4.30 TAVLTree.FindHighest . . . 188
4.4.31 TAVLTree.FindNearest . . . 188
4.4.32 TAVLTree.FindPointer . . . 188
4.4.33 TAVLTree.FindLeftMost . . . 189
4.4.34 TAVLTree.FindRightMost . . . 189
4.4.35 TAVLTree.FindLeftMostKey . . . 189
4.4.36 TAVLTree.FindRightMostKey . . . 189
4.4.37 TAVLTree.FindLeftMostSameKey . . . 190

4.4.38 TAVLTree.FindRightMostSameKey . . . 190
4.4.39 TAVLTree.GetEnumerator . . . 190
4.4.40 TAVLTree.GetEnumeratorHighToLow . . . 190
4.4.41 TAVLTree.ConsistencyCheck . . . 191
4.4.42 TAVLTree.WriteReportToStream . . . 191
4.4.43 TAVLTree.NodeToReportStr . . . 191
4.4.44 TAVLTree.ReportAsString . . . 191
4.4.45 TAVLTree.OnCompare . . . 192
4.4.46 TAVLTree.OnObjectCompare . . . 192
4.4.47 TAVLTree.NodeClass . . . 192
4.4.48 TAVLTree.Root . . . 192
4.4.49 TAVLTree.Count . . . 193
4.5 TAVLTreeNode . . . 193
4.5.1 Description . . . 193
4.5.2 Method overview . . . 193
4.5.3 TAVLTreeNode.Successor . . . 193
4.5.4 TAVLTreeNode.Precessor . . . 193
4.5.5 TAVLTreeNode.Clear . . . 194
4.5.6 TAVLTreeNode.TreeDepth . . . 194
4.5.7 TAVLTreeNode.ConsistencyCheck . . . 194
4.5.8 TAVLTreeNode.GetCount . . . 194
4.6 TAVLTreeNodeEnumerator . . . 194
4.6.1 Description . . . 194
4.6.2 Method overview . . . 195
4.6.3 Property overview . . . 195
4.6.4 TAVLTreeNodeEnumerator.Create . . . 195
4.6.5 TAVLTreeNodeEnumerator.GetEnumerator . . . 195
4.6.6 TAVLTreeNodeEnumerator.MoveNext . . . 195
4.6.7 TAVLTreeNodeEnumerator.Current . . . 195
4.6.8 TAVLTreeNodeEnumerator.LowToHigh . . . 196
4.7 TAVLTreeNodeMemManager . . . 196
4.7.1 Description . . . 196
4.7.2 Method overview . . . 196
4.7.3 Property overview . . . 196
4.7.4 TAVLTreeNodeMemManager.DisposeNode . . . 196
4.7.5 TAVLTreeNodeMemManager.NewNode . . . 197
4.7.6 TAVLTreeNodeMemManager.Clear . . . 197
4.7.7 TAVLTreeNodeMemManager.Create . . . 197
4.7.8 TAVLTreeNodeMemManager.Destroy . . . 197
4.7.9 TAVLTreeNodeMemManager.MinimumFreeNode . . . 198

4.7.10 TAVLTreeNodeMemManager.MaximumFreeNodeRatio . . . 198
4.7.11 TAVLTreeNodeMemManager.Count . . . 198
4.8 TBaseAVLTreeNodeManager . . . 198
4.8.1 Description . . . 198
4.8.2 Method overview . . . 199
4.8.3 TBaseAVLTreeNodeManager.DisposeNode . . . 199
4.8.4 TBaseAVLTreeNodeManager.NewNode . . . 199

5 Reference for unit 'base64' 200
5.1 Used units . . . 200
5.2 Overview . . . 200
5.3 Constants, types and variables . . . 200
5.3.1 Types . . . 200
5.4 Procedures and functions . . . 201
5.4.1 DecodeStringBase64 . . . 201
5.4.2 EncodeStringBase64 . . . 201
5.5 EBase64DecodingException . . . 201
5.5.1 Description . . . 201
5.6 TBase64DecodingStream . . . 202
5.6.1 Description . . . 202
5.6.2 Method overview . . . 202
5.6.3 Property overview . . . 202
5.6.4 TBase64DecodingStream.Create . . . 202
5.6.5 TBase64DecodingStream.Reset . . . 202
5.6.6 TBase64DecodingStream.Read . . . 203
5.6.7 TBase64DecodingStream.Seek . . . 203
5.6.8 TBase64DecodingStream.EOF . . . 203
5.6.9 TBase64DecodingStream.Mode . . . 203
5.7 TBase64EncodingStream . . . 204
5.7.1 Description . . . 204
5.7.2 Method overview . . . 204
5.7.3 TBase64EncodingStream.Destroy . . . 204
5.7.4 TBase64EncodingStream.Flush . . . 204
5.7.5 TBase64EncodingStream.Write . . . 205
5.7.6 TBase64EncodingStream.Seek . . . 205

6 Reference for unit 'BlowFish' 206
6.1 Used units . . . 206
6.2 Overview . . . 206
6.3 Constants, types and variables . . . 206
6.3.1 Constants . . . 206

6.3.2 Types . . . 206
6.4 EBlowFishError . . . 207
6.4.1 Description . . . 207
6.5 TBlowFish . . . 207
6.5.1 Description . . . 207
6.5.2 Method overview . . . 207
6.5.3 TBlowFish.Create . . . 207
6.5.4 TBlowFish.Encrypt . . . 208
6.5.5 TBlowFish.Decrypt . . . 208
6.6 TBlowFishDeCryptStream . . . 208
6.6.1 Description . . . 208
6.6.2 Method overview . . . 208
6.6.3 TBlowFishDeCryptStream.Create . . . 208
6.6.4 TBlowFishDeCryptStream.Read . . . 209
6.6.5 TBlowFishDeCryptStream.Seek . . . 209
6.7 TBlowFishEncryptStream . . . 209
6.7.1 Description . . . 209
6.7.2 Method overview . . . 210
6.7.3 TBlowFishEncryptStream.Destroy . . . 210
6.7.4 TBlowFishEncryptStream.Write . . . 210
6.7.5 TBlowFishEncryptStream.Seek . . . 210
6.7.6 TBlowFishEncryptStream.Flush . . . 211
6.8 TBlowFishStream . . . 211
6.8.1 Description . . . 211
6.8.2 Method overview . . . 211
6.8.3 Property overview . . . 211
6.8.4 TBlowFishStream.Create . . . 211
6.8.5 TBlowFishStream.Destroy . . . 212
6.8.6 TBlowFishStream.BlowFish . . . 212

7 Reference for unit 'BufDataset' 213
7.1 Used units . . . 213
7.2 Overview . . . 213
7.3 Constants, types and variables . . . 213
7.3.1 Types . . . 213
7.4 Procedures and functions . . . 216
7.4.1 RegisterDatapacketReader . . . 216
7.5 TApplyRecUpdateResult . . . 216
7.6 TBlobBuffer . . . 217
7.7 TBufBlobField . . . 217

7.8 TBufBookmark . . . 217
7.9 TBufRecLinkItem . . . 217
7.10 TDBCompareRec . . . 218
7.11 TRecUpdateBuffer . . . 218
7.12 TArrayBufIndex . . . 218
7.12.1 Description . . . 218
7.12.2 Method overview . . . 219
7.12.3 TArrayBufIndex.Create . . . 219
7.12.4 TArrayBufIndex.ScrollBackward . . . 219
7.12.5 TArrayBufIndex.ScrollForward . . . 219
7.12.6 TArrayBufIndex.GetCurrent . . . 219
7.12.7 TArrayBufIndex.ScrollFirst . . . 220
7.12.8 TArrayBufIndex.ScrollLast . . . 220
7.12.9 TArrayBufIndex.SetToFirstRecord . . . 220
7.12.10 TArrayBufIndex.SetToLastRecord . . . 220
7.12.11 TArrayBufIndex.StoreCurrentRecord . . . 220
7.12.12 TArrayBufIndex.RestoreCurrentRecord . . . 220
7.12.13 TArrayBufIndex.CanScrollForward . . . 221
7.12.14 TArrayBufIndex.DoScrollForward . . . 221
7.12.15 TArrayBufIndex.StoreCurrentRecIntoBookmark . . . 221
7.12.16 TArrayBufIndex.StoreSpareRecIntoBookmark . . . 221
7.12.17 TArrayBufIndex.GotoBookmark . . . 221
7.12.18 TArrayBufIndex.InitialiseIndex . . . 221
7.12.19 TArrayBufIndex.InitialiseSpareRecord . . . 221
7.12.20 TArrayBufIndex.ReleaseSpareRecord . . . 222
7.12.21 TArrayBufIndex.BeginUpdate . . . 222
7.12.22 TArrayBufIndex.AddRecord . . . 222
7.12.23 TArrayBufIndex.InsertRecordBeforeCurrentRecord . . . 222
7.12.24 TArrayBufIndex.RemoveRecordFromIndex . . . 222
7.12.25 TArrayBufIndex.EndUpdate . . . 222
7.13 TBufBlobStream . . . 223
7.13.1 Description . . . 223
7.13.2 Method overview . . . 223
7.13.3 TBufBlobStream.Create . . . 223
7.13.4 TBufBlobStream.Destroy . . . 223
7.14 TBufDataset . . . 223
7.14.1 Description . . . 223
7.14.2 Property overview . . . 224
7.14.3 TBufDataset.MaxIndexesCount . . . 224
7.14.4 TBufDataset.FieldDefs . . . 224

7.14.5 TBufDataset.Active . . . 225
7.14.6 TBufDataset.AutoCalcFields . . . 225
7.14.7 TBufDataset.Filter . . . 225
7.14.8 TBufDataset.Filtered . . . 225
7.14.9 TBufDataset.ReadOnly . . . 225
7.14.10 TBufDataset.AfterCancel . . . 225
7.14.11 TBufDataset.AfterClose . . . 226
7.14.12 TBufDataset.AfterDelete . . . 226
7.14.13 TBufDataset.AfterEdit . . . 226
7.14.14 TBufDataset.AfterInsert . . . 226
7.14.15 TBufDataset.AfterOpen . . . 226
7.14.16 TBufDataset.AfterPost . . . 226
7.14.17 TBufDataset.AfterScroll . . . 226
7.14.18 TBufDataset.BeforeCancel . . . 227
7.14.19 TBufDataset.BeforeClose . . . 227
7.14.20 TBufDataset.BeforeDelete . . . 227
7.14.21 TBufDataset.BeforeEdit . . . 227
7.14.22 TBufDataset.BeforeInsert . . . 227
7.14.23 TBufDataset.BeforeOpen . . . 227
7.14.24 TBufDataset.BeforePost . . . 227
7.14.25 TBufDataset.BeforeScroll . . . 228
7.14.26 TBufDataset.OnCalcFields . . . 228
7.14.27 TBufDataset.OnDeleteError . . . 228
7.14.28 TBufDataset.OnEditError . . . 228
7.14.29 TBufDataset.OnFilterRecord . . . 228
7.14.30 TBufDataset.OnNewRecord . . . 228
7.14.31 TBufDataset.OnPostError . . . 228
7.14.32 TBufDataset.CancelChangesOnRefresh . . . 229
7.15 TBufIndex . . . 229
7.15.1 Description . . . 229
7.15.2 Method overview . . . 230
7.15.3 Property overview . . . 230
7.15.4 TBufIndex.Create . . . 231
7.15.5 TBufIndex.ScrollBackward . . . 231
7.15.6 TBufIndex.ScrollForward . . . 231
7.15.7 TBufIndex.GetCurrent . . . 231
7.15.8 TBufIndex.ScrollFirst . . . 231
7.15.9 TBufIndex.ScrollLast . . . 231
7.15.10 TBufIndex.GetRecord . . . 232
7.15.11 TBufIndex.SetToFirstRecord . . . 232

7.15.12 TBufIndex.SetToLastRecord . . . 232
7.15.13 TBufIndex.StoreCurrentRecord . . . 232
7.15.14 TBufIndex.RestoreCurrentRecord . . . 232
7.15.15 TBufIndex.CanScrollForward . . . 232
7.15.16 TBufIndex.DoScrollForward . . . 233
7.15.17 TBufIndex.StoreCurrentRecIntoBookmark . . . 233
7.15.18 TBufIndex.StoreSpareRecIntoBookmark . . . 233
7.15.19 TBufIndex.GotoBookmark . . . 233
7.15.20 TBufIndex.BookmarkValid . . . 233
7.15.21 TBufIndex.CompareBookmarks . . . 233
7.15.22 TBufIndex.SameBookmarks . . . 234
7.15.23 TBufIndex.InitialiseIndex . . . 234
7.15.24 TBufIndex.InitialiseSpareRecord . . . 234
7.15.25 TBufIndex.ReleaseSpareRecord . . . 234
7.15.26 TBufIndex.BeginUpdate . . . 234
7.15.27 TBufIndex.AddRecord . . . 235
7.15.28 TBufIndex.InsertRecordBeforeCurrentRecord . . . 235
7.15.29 TBufIndex.RemoveRecordFromIndex . . . 235
7.15.30 TBufIndex.OrderCurrentRecord . . . 235
7.15.31 TBufIndex.EndUpdate . . . 235
7.15.32 TBufIndex.SpareRecord . . . 236
7.15.33 TBufIndex.SpareBuffer . . . 236
7.15.34 TBufIndex.CurrentRecord . . . 236
7.15.35 TBufIndex.CurrentBuffer . . . 236
7.15.36 TBufIndex.IsInitialized . . . 236
7.15.37 TBufIndex.BookmarkSize . . . 237
7.15.38 TBufIndex.RecNo . . . 237
7.16 TCustomBufDataset . . . 237
7.16.1 Description . . . 237
7.16.2 Method overview . . . 238
7.16.3 Property overview . . . 239
7.16.4 TCustomBufDataset.Create . . . 239
7.16.5 TCustomBufDataset.GetFieldData . . . 240
7.16.6 TCustomBufDataset.SetFieldData . . . 240
7.16.7 TCustomBufDataset.ApplyUpdates . . . 240
7.16.8 TCustomBufDataset.MergeChangeLog . . . 241
7.16.9 TCustomBufDataset.RevertRecord . . . 241
7.16.10 TCustomBufDataset.CancelUpdates . . . 241
7.16.11 TCustomBufDataset.Destroy . . . 241
7.16.12 TCustomBufDataset.Locate . . . 242

7.16.13 TCustomBufDataset.Lookup . . . 242
7.16.14 TCustomBufDataset.UpdateStatus . . . 243
7.16.15 TCustomBufDataset.CreateBlobStream . . . 244
7.16.16 TCustomBufDataset.AddIndex . . . 244
7.16.17 TCustomBufDataset.ClearIndexes . . . 245
7.16.18 TCustomBufDataset.SetDatasetPacket . . . 245
7.16.19 TCustomBufDataset.GetDatasetPacket . . . 245
7.16.20 TCustomBufDataset.LoadFromStream . . . 246
7.16.21 TCustomBufDataset.SaveToStream . . . 246
7.16.22 TCustomBufDataset.LoadFromFile . . . 247
7.16.23 TCustomBufDataset.SaveToFile . . . 247
7.16.24 TCustomBufDataset.CreateDataset . . . 248
7.16.25 TCustomBufDataset.Clear . . . 248
7.16.26 TCustomBufDataset.BookmarkValid . . . 249
7.16.27 TCustomBufDataset.CompareBookmarks . . . 249
7.16.28 TCustomBufDataset.CopyFromDataset . . . 249
7.16.29 TCustomBufDataset.ChangeCount . . . 250
7.16.30 TCustomBufDataset.MaxIndexesCount . . . 250
7.16.31 TCustomBufDataset.ReadOnly . . . 251
7.16.32 TCustomBufDataset.ManualMergeChangeLog . . . 251
7.16.33 TCustomBufDataset.FileName . . . 251
7.16.34 TCustomBufDataset.PacketRecords . . . 252
7.16.35 TCustomBufDataset.OnUpdateError . . . 252
7.16.36 TCustomBufDataset.IndexDefs . . . 253
7.16.37 TCustomBufDataset.IndexName . . . 253
7.16.38 TCustomBufDataset.IndexFieldNames . . . 253
7.16.39 TCustomBufDataset.UniDirectional . . . 254
7.17 TDataPacketHandler . . . 254
7.17.1 Method overview . . . 254
7.17.2 TDataPacketHandler.Create . . . 255
7.17.3 TDataPacketHandler.LoadFieldDefs . . . 255
7.17.4 TDataPacketHandler.InitLoadRecords . . . 255
7.17.5 TDataPacketHandler.GetCurrentRecord . . . 255
7.17.6 TDataPacketHandler.GetRecordRowState . . . 255
7.17.7 TDataPacketHandler.RestoreRecord . . . 255
7.17.8 TDataPacketHandler.GotoNextRecord . . . 255
7.17.9 TDataPacketHandler.StoreFieldDefs . . . 255
7.17.10 TDataPacketHandler.StoreRecord . . . 256
7.17.11 TDataPacketHandler.FinalizeStoreRecords . . . 256
7.17.12 TDataPacketHandler.RecognizeStream . . . 256

7.18 TDoubleLinkedBufIndex . . . 256
7.18.1 Description . . . 256
7.18.2 Method overview . . . 257
7.18.3 TDoubleLinkedBufIndex.ScrollBackward . . . 257
7.18.4 TDoubleLinkedBufIndex.ScrollForward . . . 257
7.18.5 TDoubleLinkedBufIndex.GetCurrent . . . 257
7.18.6 TDoubleLinkedBufIndex.ScrollFirst . . . 258
7.18.7 TDoubleLinkedBufIndex.ScrollLast . . . 258
7.18.8 TDoubleLinkedBufIndex.GetRecord . . . 258
7.18.9 TDoubleLinkedBufIndex.SetToFirstRecord . . . 258
7.18.10 TDoubleLinkedBufIndex.SetToLastRecord . . . 258
7.18.11 TDoubleLinkedBufIndex.StoreCurrentRecord . . . 258
7.18.12 TDoubleLinkedBufIndex.RestoreCurrentRecord . . . 259
7.18.13 TDoubleLinkedBufIndex.CanScrollForward . . . 259
7.18.14 TDoubleLinkedBufIndex.DoScrollForward . . . 259
7.18.15 TDoubleLinkedBufIndex.StoreCurrentRecIntoBookmark . . . 259
7.18.16 TDoubleLinkedBufIndex.StoreSpareRecIntoBookmark . . . 259
7.18.17 TDoubleLinkedBufIndex.GotoBookmark . . . 259
7.18.18 TDoubleLinkedBufIndex.CompareBookmarks . . . 260
7.18.19 TDoubleLinkedBufIndex.SameBookmarks . . . 260
7.18.20 TDoubleLinkedBufIndex.InitialiseIndex . . . 260
7.18.21 TDoubleLinkedBufIndex.InitialiseSpareRecord . . . 260
7.18.22 TDoubleLinkedBufIndex.ReleaseSpareRecord . . . 260
7.18.23 TDoubleLinkedBufIndex.BeginUpdate . . . 260
7.18.24 TDoubleLinkedBufIndex.AddRecord . . . 261
7.18.25 TDoubleLinkedBufIndex.InsertRecordBeforeCurrentRecord . . . 261
7.18.26 TDoubleLinkedBufIndex.RemoveRecordFromIndex . . . 261
7.18.27 TDoubleLinkedBufIndex.OrderCurrentRecord . . . 261
7.18.28 TDoubleLinkedBufIndex.EndUpdate . . . 261
7.19 TFpcBinaryDatapacketHandler . . . 262
7.19.1 Method overview . . . 262
7.19.2 TFpcBinaryDatapacketHandler.Create . . . 262
7.19.3 TFpcBinaryDatapacketHandler.LoadFieldDefs . . . 262
7.19.4 TFpcBinaryDatapacketHandler.StoreFieldDefs . . . 262
7.19.5 TFpcBinaryDatapacketHandler.InitLoadRecords . . . 262
7.19.6 TFpcBinaryDatapacketHandler.GetCurrentRecord . . . 262
7.19.7 TFpcBinaryDatapacketHandler.GetRecordRowState . . . 262
7.19.8 TFpcBinaryDatapacketHandler.RestoreRecord . . . 263
7.19.9 TFpcBinaryDatapacketHandler.GotoNextRecord . . . 263
7.19.10 TFpcBinaryDatapacketHandler.StoreRecord . . . 263

7.19.11 TFpcBinaryDatapacketHandler.FinalizeStoreRecords . . . 263
7.19.12 TFpcBinaryDatapacketHandler.RecognizeStream . . . 263
7.20 TUniDirectionalBufIndex . . . 263
7.20.1 Description . . . 263
7.20.2 Method overview . . . 264
7.20.3 TUniDirectionalBufIndex.ScrollBackward . . . 264
7.20.4 TUniDirectionalBufIndex.ScrollForward . . . 264
7.20.5 TUniDirectionalBufIndex.GetCurrent . . . 264
7.20.6 TUniDirectionalBufIndex.ScrollFirst . . . 264
7.20.7 TUniDirectionalBufIndex.ScrollLast . . . 265
7.20.8 TUniDirectionalBufIndex.SetToFirstRecord . . . 265
7.20.9 TUniDirectionalBufIndex.SetToLastRecord . . . 265
7.20.10 TUniDirectionalBufIndex.StoreCurrentRecord . . . 265
7.20.11 TUniDirectionalBufIndex.RestoreCurrentRecord . . . 265
7.20.12 TUniDirectionalBufIndex.CanScrollForward . . . 265
7.20.13 TUniDirectionalBufIndex.DoScrollForward . . . 266
7.20.14 TUniDirectionalBufIndex.StoreCurrentRecIntoBookmark . . . 266
7.20.15 TUniDirectionalBufIndex.StoreSpareRecIntoBookmark . . . 266
7.20.16 TUniDirectionalBufIndex.GotoBookmark . . . 266
7.20.17 TUniDirectionalBufIndex.InitialiseIndex . . . 266
7.20.18 TUniDirectionalBufIndex.InitialiseSpareRecord . . . 266
7.20.19 TUniDirectionalBufIndex.ReleaseSpareRecord . . . 266
7.20.20 TUniDirectionalBufIndex.BeginUpdate . . . 267
7.20.21 TUniDirectionalBufIndex.AddRecord . . . 267
7.20.22 TUniDirectionalBufIndex.InsertRecordBeforeCurrentRecord . . . 267
7.20.23 TUniDirectionalBufIndex.RemoveRecordFromIndex . . . 267
7.20.24 TUniDirectionalBufIndex.OrderCurrentRecord . . . 267
7.20.25 TUniDirectionalBufIndex.EndUpdate . . . 267

8 Reference for unit 'bufstream' 268
8.1 Used units . . . 268
8.2 Overview . . . 268
8.3 Constants, types and variables . . . 268
8.3.1 Constants . . . 268
8.4 TBufferedFileStream . . . 269
8.4.1 Description . . . 269
8.4.2 Method overview . . . 269
8.4.3 TBufferedFileStream.Create . . . 269
8.4.4 TBufferedFileStream.Destroy . . . 270
8.4.5 TBufferedFileStream.Seek . . . 270

8.4.6 TBufferedFileStream.Read . . . 271
8.4.7 TBufferedFileStream.Write . . . 271
8.4.8 TBufferedFileStream.Flush . . . 272
8.4.9 TBufferedFileStream.InitializeCache . . . 272
8.5 TBufStream . . . 272
8.5.1 Description . . . 272
8.5.2 Method overview . . . 273
8.5.3 Property overview . . . 273
8.5.4 TBufStream.Create . . . 273
8.5.5 TBufStream.Destroy . . . 273
8.5.6 TBufStream.Buffer . . . 273
8.5.7 TBufStream.Capacity . . . 274
8.5.8 TBufStream.BufferPos . . . 274
8.5.9 TBufStream.BufferSize . . . 274
8.6 TReadBufStream . . . 274
8.6.1 Description . . . 274
8.6.2 Method overview . . . 275
8.6.3 TReadBufStream.Seek . . . 275
8.6.4 TReadBufStream.Read . . . 275
8.7 TWriteBufStream . . . 275
8.7.1 Description . . . 275
8.7.2 Method overview . . . 275
8.7.3 TWriteBufStream.Destroy . . . 276
8.7.4 TWriteBufStream.Seek . . . 276
8.7.5 TWriteBufStream.Write . . . 276

9 Reference for unit 'CacheCls' 277
9.1 Used units . . . 277
9.2 Overview . . . 277
9.3 Constants, types and variables . . . 277
9.3.1 Resource strings . . . 277
9.3.2 Types . . . 277
9.4 TCacheSlot . . . 278
9.5 ECacheError . . . 278
9.5.1 Description . . . 278
9.6 TCache . . . 278
9.6.1 Description . . . 278
9.6.2 Method overview . . . 279
9.6.3 Property overview . . . 279
9.6.4 TCache.Create . . . 279

9.6.5 TCache.Destroy . . . 279
9.6.6 TCache.Add . . . 279
9.6.7 TCache.AddNew . . . 280
9.6.8 TCache.FindSlot . . . 280
9.6.9 TCache.IndexOf . . . 280
9.6.10 TCache.Remove . . . 281
9.6.11 TCache.Data . . . 281
9.6.12 TCache.MRUSlot . . . 281
9.6.13 TCache.LRUSlot . . . 282
9.6.14 TCache.SlotCount . . . 282
9.6.15 TCache.Slots . . . 282
9.6.16 TCache.OnIsDataEqual . . . 282
9.6.17 TCache.OnFreeSlot . . . 283

10 Reference for unit 'Contnrs' 284
10.1 Used units . . . 284
10.2 Overview . . . 284
10.3 Constants, types and variables . . . 284
10.3.1 Constants . . . 284
10.3.2 Types . . . 285
10.4 Procedures and functions . . . 288
10.4.1 RSHash . . . 288
10.4.2 ViDataSize . . . 288
10.4.3 ViGet . . . 288
10.4.4 ViSet . . . 288
10.5 TBucket . . . 288
10.6 TBucketItem . . . 289
10.7 TFPHashListItem . . . 289
10.8 EDuplicate . . . 289
10.8.1 Description . . . 289
10.9 EKeyNotFound . . . 289
10.9.1 Description . . . 289
10.10 TBucketList . . . 289
10.10.1 Description . . . 289
10.10.2 Method overview . . . 290
10.10.3 TBucketList.Create . . . 290
10.11 TClassList . . . 290
10.11.1 Description . . . 290
10.11.2 Method overview . . . 290
10.11.3 Property overview . . . 290

10.11.4 TClassList.Add . . . 291
10.11.5 TClassList.Extract . . . 291
10.11.6 TClassList.Remove . . . 291
10.11.7 TClassList.IndexOf . . . 291
10.11.8 TClassList.First . . . 292
10.11.9 TClassList.Last . . . 292
10.11.10TClassList.Insert . . . 292
10.11.11TClassList.Items . . . 292
10.12TComponentList . . . 293
10.12.1 Description . . . 293
10.12.2 Method overview . . . 293
10.12.3 Property overview . . . 293
10.12.4 TComponentList.Destroy . . . 293
10.12.5 TComponentList.Add . . . 293
10.12.6 TComponentList.Extract . . . 294
10.12.7 TComponentList.Remove . . . 294
10.12.8 TComponentList.IndexOf . . . 294
10.12.9 TComponentList.First . . . 295
10.12.10TComponentList.Last . . . 295
10.12.11TComponentList.Insert . . . 295
10.12.12TComponentList.Items . . . 295
10.13TCustomBucketList . . . 296
10.13.1 Description . . . 296
10.13.2 Method overview . . . 296
10.13.3 Property overview . . . 296
10.13.4 TCustomBucketList.Destroy . . . 296
10.13.5 TCustomBucketList.Clear . . . 296
10.13.6 TCustomBucketList.Add . . . 297
10.13.7 TCustomBucketList.Assign . . . 297
10.13.8 TCustomBucketList.Exists . . . 297
10.13.9 TCustomBucketList.Find . . . 297
10.13.10TCustomBucketList.ForEach . . . 298
10.13.11TCustomBucketList.Remove . . . 298
10.13.12TCustomBucketList.Data . . . 298
10.14TFPCustomHashTable . . . 298
10.14.1 Description . . . 298
10.14.2 Method overview . . . 299
10.14.3 Property overview . . . 299
10.14.4 TFPCustomHashTable.Create . . . 299
10.14.5 TFPCustomHashTable.CreateWith . . . 300

10.14.6 TFPCustomHashTable.Destroy . . . 300
10.14.7 TFPCustomHashTable.ChangeTableSize . . . 300
10.14.8 TFPCustomHashTable.Clear . . . 300
10.14.9 TFPCustomHashTable.Delete . . . 301
10.14.10TFPCustomHashTable.Find . . . 301
10.14.11TFPCustomHashTable.IsEmpty . . . 301
10.14.12TFPCustomHashTable.HashFunction . . . 301
10.14.13TFPCustomHashTable.Count . . . 302
10.14.14TFPCustomHashTable.HashTableSize . . . 302
10.14.15TFPCustomHashTable.HashTable . . . 302
10.14.16TFPCustomHashTable.VoidSlots . . . 302
10.14.17TFPCustomHashTable.LoadFactor . . . 303
10.14.18TFPCustomHashTable.AVGChainLen . . . 303
10.14.19TFPCustomHashTable.MaxChainLength . . . 303
10.14.20TFPCustomHashTable.NumberOfCollisions . . . 303
10.14.21TFPCustomHashTable.Density . . . 304
10.15TFPDataHashTable . . . 304
10.15.1 Description . . . 304
10.15.2 Method overview . . . 304
10.15.3 Property overview . . . 304
10.15.4 TFPDataHashTable.Iterate . . . 304
10.15.5 TFPDataHashTable.Add . . . 305
10.15.6 TFPDataHashTable.Items . . . 305
10.16TFPHashList . . . 305
10.16.1 Description . . . 305
10.16.2 Method overview . . . 306
10.16.3 Property overview . . . 306
10.16.4 TFPHashList.Create . . . 306
10.16.5 TFPHashList.Destroy . . . 306
10.16.6 TFPHashList.Add . . . 307
10.16.7 TFPHashList.Clear . . . 307
10.16.8 TFPHashList.NameOfIndex . . . 307
10.16.9 TFPHashList.HashOfIndex . . . 307
10.16.10TFPHashList.GetNextCollision . . . 308
10.16.11TFPHashList.Delete . . . 308
10.16.12TFPHashList.Error . . . 308
10.16.13TFPHashList.Extract . . . 308
10.16.14TFPHashList.IndexOf . . . 308
10.16.15TFPHashList.Find . . . 309
10.16.16TFPHashList.FindIndexOf . . . 309

10.16.17TFPHashList.FindWithHash . . . 309
10.16.18TFPHashList.Rename . . . 310
10.16.19TFPHashList.Remove . . . 310
10.16.20TFPHashList.Pack . . . 310
10.16.21TFPHashList.ForEachCall . . . 310
10.16.22TFPHashList.Count . . . 310
10.16.23TFPHashList.Capacity . . . 311
10.16.24TFPHashList.Items . . . 311
10.16.25TFPHashList.List . . . 311
10.17TFPHashObject . . . 312
10.17.1 Description . . . 312
10.17.2 Method overview . . . 312
10.17.3 Property overview . . . 312
10.17.4 TFPHashObject.CreateNotOwned . . . 312
10.17.5 TFPHashObject.Create . . . 312
10.17.6 TFPHashObject.ChangeOwner . . . 313
10.17.7 TFPHashObject.ChangeOwnerAndName . . . 313
10.17.8 TFPHashObject.Rename . . . 313
10.17.9 TFPHashObject.Name . . . 313
10.17.10TFPHashObject.Hash . . . 314
10.17.11TFPHashObject.OwnerList . . . 314
10.18TFPHashObjectList . . . 314
10.18.1 Method overview . . . 314
10.18.2 Property overview . . . 314
10.18.3 TFPHashObjectList.Create . . . 315
10.18.4 TFPHashObjectList.Destroy . . . 315
10.18.5 TFPHashObjectList.Clear . . . 315
10.18.6 TFPHashObjectList.Add . . . 315
10.18.7 TFPHashObjectList.NameOfIndex . . . 316
10.18.8 TFPHashObjectList.HashOfIndex . . . 316
10.18.9 TFPHashObjectList.GetNextCollision . . . 316
10.18.10TFPHashObjectList.Delete . . . 316
10.18.11TFPHashObjectList.Extract . . . 317
10.18.12TFPHashObjectList.Remove . . . 317
10.18.13TFPHashObjectList.IndexOf . . . 317
10.18.14TFPHashObjectList.Find . . . 317
10.18.15TFPHashObjectList.FindIndexOf . . . 318
10.18.16TFPHashObjectList.FindWithHash . . . 318
10.18.17TFPHashObjectList.Rename . . . 318
10.18.18TFPHashObjectList.FindInstanceOf . . . 318

10.18.19TFPHashObjectList.Pack . . . 319
10.18.20TFPHashObjectList.ForEachCall . . . 319
10.18.21TFPHashObjectList.Capacity . . . 319
10.18.22TFPHashObjectList.Count . . . 319
10.18.23TFPHashObjectList.OwnsObjects . . . 320
10.18.24TFPHashObjectList.Items . . . 320
10.18.25TFPHashObjectList.List . . . 320
10.19TFPObjectHashTable . . . 320
10.19.1 Description . . . 320
10.19.2 Method overview . . . 321
10.19.3 Property overview . . . 321
10.19.4 TFPObjectHashTable.Create . . . 321
10.19.5 TFPObjectHashTable.CreateWith . . . 321
10.19.6 TFPObjectHashTable.Iterate . . . 322
10.19.7 TFPObjectHashTable.Add . . . 322
10.19.8 TFPObjectHashTable.Items . . . 322
10.19.9 TFPObjectHashTable.OwnsObjects . . . 322
10.20TFPObjectList . . . 323
10.20.1 Description . . . 323
10.20.2 Method overview . . . 323
10.20.3 Property overview . . . 323
10.20.4 TFPObjectList.Create . . . 323
10.20.5 TFPObjectList.Destroy . . . 324
10.20.6 TFPObjectList.Clear . . . 324
10.20.7 TFPObjectList.Add . . . 324
10.20.8 TFPObjectList.Delete . . . 324
10.20.9 TFPObjectList.Exchange . . . 325
10.20.10TFPObjectList.Expand . . . 325
10.20.11TFPObjectList.Extract . . . 325
10.20.12TFPObjectList.Remove . . . 326
10.20.13TFPObjectList.IndexOf . . . 326
10.20.14TFPObjectList.FindInstanceOf . . . 326
10.20.15TFPObjectList.Insert . . . 327
10.20.16TFPObjectList.First . . . 327
10.20.17TFPObjectList.Last . . . 327
10.20.18TFPObjectList.Move . . . 327
10.20.19TFPObjectList.Assign . . . 328
10.20.20TFPObjectList.Pack . . . 328
10.20.21TFPObjectList.Sort . . . 328
10.20.22TFPObjectList.ForEachCall . . . 329

10.20.23TFPObjectList.Capacity . . . 329
10.20.24TFPObjectList.Count . . . 329
10.20.25TFPObjectList.OwnsObjects . . . 329
10.20.26TFPObjectList.Items . . . 330
10.20.27TFPObjectList.List . . . 330
10.21TFPStringHashTable . . . 330
10.21.1 Description . . . 330
10.21.2 Method overview . . . 330
10.21.3 Property overview . . . 331
10.21.4 TFPStringHashTable.Iterate . . . 331
10.21.5 TFPStringHashTable.Add . . . 331
10.21.6 TFPStringHashTable.Items . . . 331
10.22THTCustomNode . . . 331
10.22.1 Description . . . 331
10.22.2 Method overview . . . 332
10.22.3 Property overview . . . 332
10.22.4 THTCustomNode.CreateWith . . . 332
10.22.5 THTCustomNode.HasKey . . . 332
10.22.6 THTCustomNode.Key . . . 332
10.23THTDataNode . . . 333
10.23.1 Description . . . 333
10.23.2 Property overview . . . 333
10.23.3 THTDataNode.Data . . . 333
10.24THTObjectNode . . . 333
10.24.1 Description . . . 333
10.24.2 Property overview . . . 333
10.24.3 THTObjectNode.Data . . . 333
10.25THTOwnedObjectNode . . . 334
10.25.1 Description . . . 334
10.25.2 Method overview . . . 334
10.25.3 THTOwnedObjectNode.Destroy . . . 334
10.26THTStringNode . . . 334
10.26.1 Description . . . 334
10.26.2 Property overview . . . 334
10.26.3 THTStringNode.Data . . . 334
10.27TObjectBucketList . . . 335
10.27.1 Description . . . 335
10.27.2 Method overview . . . 335
10.27.3 Property overview . . . 335
10.27.4 TObjectBucketList.Add . . . 335

10.27.5 TObjectBucketList.Remove . . . 335
10.27.6 TObjectBucketList.Data . . . 336
10.28 TObjectList . . . 336
10.28.1 Description . . . 336
10.28.2 Method overview . . . 336
10.28.3 Property overview . . . 336
10.28.4 TObjectList.Create . . . 336
10.28.5 TObjectList.Add . . . 337
10.28.6 TObjectList.Extract . . . 337
10.28.7 TObjectList.Remove . . . 337
10.28.8 TObjectList.IndexOf . . . 338
10.28.9 TObjectList.FindInstanceOf . . . 338
10.28.10 TObjectList.Insert . . . 338
10.28.11 TObjectList.First . . . 338
10.28.12 TObjectList.Last . . . 339
10.28.13 TObjectList.OwnsObjects . . . 339
10.28.14 TObjectList.Items . . . 339
10.29 TObjectQueue . . . 339
10.29.1 Method overview . . . 339
10.29.2 TObjectQueue.Push . . . 340
10.29.3 TObjectQueue.Pop . . . 340
10.29.4 TObjectQueue.Peek . . . 340
10.30 TObjectStack . . . 340
10.30.1 Description . . . 340
10.30.2 Method overview . . . 341
10.30.3 TObjectStack.Push . . . 341
10.30.4 TObjectStack.Pop . . . 341
10.30.5 TObjectStack.Peek . . . 341
10.31 TOrderedList . . . 342
10.31.1 Description . . . 342
10.31.2 Method overview . . . 342
10.31.3 TOrderedList.Create . . . 342
10.31.4 TOrderedList.Destroy . . . 342
10.31.5 TOrderedList.Count . . . 343
10.31.6 TOrderedList.AtLeast . . . 343
10.31.7 TOrderedList.Push . . . 343
10.31.8 TOrderedList.Pop . . . 343
10.31.9 TOrderedList.Peek . . . 344
10.32 TQueue . . . 344
10.32.1 Description . . . 344

10.33TStack 344
10.33.1 Description 344

11 Reference for unit 'csvdocument' 345
11.1 Used units 345
11.2 Overview 345
11.3 Constants, types and variables 345
11.3.1 Types 345
11.4 TCSVDocument 346
11.4.1 Description 346
11.4.2 Method overview 346
11.4.3 Property overview 346
11.4.4 TCSVDocument.Create 346
11.4.5 TCSVDocument.Destroy 347
11.4.6 TCSVDocument.LoadFromFile 347
11.4.7 TCSVDocument.LoadFromStream 347
11.4.8 TCSVDocument.SaveToFile 348
11.4.9 TCSVDocument.SaveToStream 348
11.4.10 TCSVDocument.AddRow 348
11.4.11 TCSVDocument.AddCell 348
11.4.12 TCSVDocument.InsertRow 349
11.4.13 TCSVDocument.InsertCell 349
11.4.14 TCSVDocument.RemoveRow 349
11.4.15 TCSVDocument.RemoveCell 349
11.4.16 TCSVDocument.HasRow 350
11.4.17 TCSVDocument.HasCell 350
11.4.18 TCSVDocument.IndexOfCol 350
11.4.19 TCSVDocument.IndexOfRow 350
11.4.20 TCSVDocument.Clear 351
11.4.21 TCSVDocument.CloneRow 351
11.4.22 TCSVDocument.ExchangeRows 351
11.4.23 TCSVDocument.UnifyEmbeddedLineEndings 351
11.4.24 TCSVDocument.RemoveTrailingEmptyCells 351
11.4.25 TCSVDocument.DetectBOM 352
11.4.26 TCSVDocument.Cells 352
11.4.27 TCSVDocument.RowCount 352
11.4.28 TCSVDocument.ColCount 352
11.4.29 TCSVDocument.MaxColCount 353
11.4.30 TCSVDocument.CSVText 353

12 Reference for unit 'csvreadwrite' 354

12.1 Used units . . . 354
12.2 Overview . . . 354
12.3 Constants, types and variables . . . 354
12.3.1 Types . . . 354
12.4 Procedures and functions . . . 355
12.4.1 ChangeLineEndings . . . 355
12.5 TCSVBuilder . . . 355
12.5.1 Description . . . 355
12.5.2 Method overview . . . 355
12.5.3 Property overview . . . 356
12.5.4 TCSVBuilder.Create . . . 356
12.5.5 TCSVBuilder.Destroy . . . 356
12.5.6 TCSVBuilder.SetOutput . . . 356
12.5.7 TCSVBuilder.ResetBuilder . . . 356
12.5.8 TCSVBuilder.AppendCell . . . 357
12.5.9 TCSVBuilder.AppendRow . . . 357
12.5.10 TCSVBuilder.DefaultOutput . . . 357
12.5.11 TCSVBuilder.DefaultOutputAsString . . . 357
12.6 TCSVHandler . . . 358
12.6.1 Description . . . 358
12.6.2 Method overview . . . 358
12.6.3 Property overview . . . 358
12.6.4 TCSVHandler.Create . . . 358
12.6.5 TCSVHandler.Assign . . . 358
12.6.6 TCSVHandler.AssignCSVProperties . . . 359
12.6.7 TCSVHandler.Delimiter . . . 359
12.6.8 TCSVHandler.QuoteChar . . . 359
12.6.9 TCSVHandler.LineEnding . . . 359
12.6.10 TCSVHandler.IgnoreOuterWhitespace . . . 360
12.6.11 TCSVHandler.QuoteOuterWhitespace . . . 360
12.6.12 TCSVHandler.EqualColCountPerRow . . . 360
12.7 TCSVParser . . . 360
12.7.1 Description . . . 360
12.7.2 Method overview . . . 361
12.7.3 Property overview . . . 361
12.7.4 TCSVParser.Create . . . 361
12.7.5 TCSVParser.Destroy . . . 361
12.7.6 TCSVParser.SetSource . . . 362
12.7.7 TCSVParser.ResetParser . . . 362
12.7.8 TCSVParser.ParseNextCell . . . 362

12.7.9 TCSVParser.CurrentRow . . . 362
12.7.10 TCSVParser.CurrentCol . . . 363
12.7.11 TCSVParser.CurrentCellText . . . 363
12.7.12 TCSVParser.MaxColCount . . . 363
12.7.13 TCSVParser.FreeStream . . . 364
12.7.14 TCSVParser.BOM . . . 364
12.7.15 TCSVParser.DetectBOM . . . 364

13 Reference for unit 'CustApp' 365
13.1 Used units . . . 365
13.2 Overview . . . 365
13.3 Constants, types and variables . . . 365
13.3.1 Types . . . 365
13.3.2 Variables . . . 366
13.4 TCustomApplication . . . 366
13.4.1 Description . . . 366
13.4.2 Method overview . . . 367
13.4.3 Property overview . . . 367
13.4.4 TCustomApplication.Create . . . 367
13.4.5 TCustomApplication.Destroy . . . 368
13.4.6 TCustomApplication.HandleException . . . 368
13.4.7 TCustomApplication.Initialize . . . 368
13.4.8 TCustomApplication.Run . . . 369
13.4.9 TCustomApplication.ShowException . . . 369
13.4.10 TCustomApplication.Terminate . . . 369
13.4.11 TCustomApplication.FindOptionIndex . . . 370
13.4.12 TCustomApplication.GetOptionValue . . . 370
13.4.13 TCustomApplication.GetOptionValues . . . 371
13.4.14 TCustomApplication.HasOption . . . 371
13.4.15 TCustomApplication.CheckOptions . . . 371
13.4.16 TCustomApplication.GetNonOptions . . . 372
13.4.17 TCustomApplication.GetEnvironmentList . . . 373
13.4.18 TCustomApplication.Log . . . 373
13.4.19 TCustomApplication.ExeName . . . 373
13.4.20 TCustomApplication.HelpFile . . . 374
13.4.21 TCustomApplication.Terminated . . . 374
13.4.22 TCustomApplication.Title . . . 374
13.4.23 TCustomApplication.OnException . . . 374
13.4.24 TCustomApplication.ConsoleApplication . . . 375
13.4.25 TCustomApplication.Location . . . 375

13.4.26 TCustomApplication.Params . . . 375
13.4.27 TCustomApplication.ParamCount . . . 376
13.4.28 TCustomApplication.EnvironmentVariable . . . 376
13.4.29 TCustomApplication.OptionChar . . . 376
13.4.30 TCustomApplication.CaseSensitiveOptions . . . 376
13.4.31 TCustomApplication.CaseSensitiveShortOptions . . . 377
13.4.32 TCustomApplication.CaseSensitiveLongOptions . . . 377
13.4.33 TCustomApplication.StopOnException . . . 377
13.4.34 TCustomApplication.ExceptionExitCode . . . 377
13.4.35 TCustomApplication.EventLogFilter . . . 378
13.4.36 TCustomApplication.SingleInstance . . . 378
13.4.37 TCustomApplication.SingleInstanceClass . . . 378
13.4.38 TCustomApplication.SingleInstanceEnabled . . . 378

14 Reference for unit 'custweb' 380
14.1 Used units . . . 380
14.2 Overview . . . 380
14.3 Constants, types and variables . . . 380
14.3.1 Types . . . 380
14.4 Procedures and functions . . . 381
14.4.1 ExceptionToHTML . . . 381
14.5 EFPWebError . . . 381
14.5.1 Description . . . 381
14.6 TCustomWebApplication . . . 382
14.6.1 Description . . . 382
14.6.2 Method overview . . . 382
14.6.3 Property overview . . . 382
14.6.4 TCustomWebApplication.Create . . . 382
14.6.5 TCustomWebApplication.Destroy . . . 383
14.6.6 TCustomWebApplication.CreateForm . . . 383
14.6.7 TCustomWebApplication.Initialize . . . 383
14.6.8 TCustomWebApplication.Terminate . . . 384
14.6.9 TCustomWebApplication.HandleGetOnPost . . . 384
14.6.10 TCustomWebApplication.RedirectOnError . . . 384
14.6.11 TCustomWebApplication.RedirectOnErrorURL . . . 384
14.6.12 TCustomWebApplication.ApplicationURL . . . 385
14.6.13 TCustomWebApplication.AllowDefaultModule . . . 385
14.6.14 TCustomWebApplication.DefaultModuleName . . . 385
14.6.15 TCustomWebApplication.ModuleVariable . . . 385
14.6.16 TCustomWebApplication.OnGetModule . . . 386

14.6.17 TCustomWebApplication.Email . . . 386
14.6.18 TCustomWebApplication.Administrator . . . 386
14.6.19 TCustomWebApplication.OnShowRequestException . . . 386
14.6.20 TCustomWebApplication.OnUnknownRequestEncoding . . . 387
14.6.21 TCustomWebApplication.EventLog . . . 387
14.6.22 TCustomWebApplication.PreferModuleName . . . 387
14.6.23 TCustomWebApplication.LegacyRouting . . . 387
14.7 TWebHandler . . . 388
14.7.1 Description . . . 388
14.7.2 Method overview . . . 388
14.7.3 Property overview . . . 389
14.7.4 TWebHandler.Create . . . 389
14.7.5 TWebHandler.Destroy . . . 389
14.7.6 TWebHandler.Run . . . 390
14.7.7 TWebHandler.Log . . . 390
14.7.8 TWebHandler.DoHandleRequest . . . 390
14.7.9 TWebHandler.HandleRequest . . . 391
14.7.10 TWebHandler.HandleGetOnPost . . . 391
14.7.11 TWebHandler.RedirectOnError . . . 391
14.7.12 TWebHandler.RedirectOnErrorURL . . . 391
14.7.13 TWebHandler.ApplicationURL . . . 392
14.7.14 TWebHandler.AllowDefaultModule . . . 392
14.7.15 TWebHandler.DefaultModuleName . . . 392
14.7.16 TWebHandler.ModuleVariable . . . 392
14.7.17 TWebHandler.OnGetModule . . . 393
14.7.18 TWebHandler.Email . . . 393
14.7.19 TWebHandler.Title . . . 393
14.7.20 TWebHandler.Administrator . . . 393
14.7.21 TWebHandler.OnShowRequestException . . . 394
14.7.22 TWebHandler.OnIdle . . . 394
14.7.23 TWebHandler.OnLog . . . 394
14.7.24 TWebHandler.OnUnknownRequestEncoding . . . 394
14.7.25 TWebHandler.OnInitModule . . . 395
14.7.26 TWebHandler.PreferModuleName . . . 395
14.7.27 TWebHandler.LegacyRouting . . . 395

15 Reference for unit 'daemonapp' 396
15.1 Used units . . . 396
15.2 Overview . . . 396
15.3 Constants, types and variables . . . 397

15.3.1 Resource strings . . . 397
15.3.2 Types . . . 398
15.3.3 Variables . . . 402
15.4 Procedures and functions . . . 402
15.4.1 Application . . . 402
15.4.2 DaemonError . . . 402
15.4.3 RegisterDaemonApplicationClass . . . 403
15.4.4 RegisterDaemonClass . . . 403
15.4.5 RegisterDaemonMapper . . . 403
15.5 EDaemon . . . 403
15.5.1 Description . . . 403
15.6 TCustomDaemon . . . 404
15.6.1 Description . . . 404
15.6.2 Method overview . . . 404
15.6.3 Property overview . . . 404
15.6.4 TCustomDaemon.CheckControlMessages . . . 404
15.6.5 TCustomDaemon.LogMessage . . . 404
15.6.6 TCustomDaemon.ReportStatus . . . 405
15.6.7 TCustomDaemon.Definition . . . 405
15.6.8 TCustomDaemon.DaemonThread . . . 405
15.6.9 TCustomDaemon.Controller . . . 406
15.6.10 TCustomDaemon.Status . . . 406
15.6.11 TCustomDaemon.Logger . . . 406
15.7 TCustomDaemonApplication . . . 406
15.7.1 Description . . . 406
15.7.2 Method overview . . . 407
15.7.3 Property overview . . . 407
15.7.4 TCustomDaemonApplication.Create . . . 407
15.7.5 TCustomDaemonApplication.Destroy . . . 407
15.7.6 TCustomDaemonApplication.ShowException . . . 407
15.7.7 TCustomDaemonApplication.CreateDaemon . . . 408
15.7.8 TCustomDaemonApplication.StopDaemons . . . 408
15.7.9 TCustomDaemonApplication.InstallDaemons . . . 408
15.7.10 TCustomDaemonApplication.RunDaemons . . . 408
15.7.11 TCustomDaemonApplication.UnInstallDaemons . . . 409
15.7.12 TCustomDaemonApplication.ShowHelp . . . 409
15.7.13 TCustomDaemonApplication.CreateForm . . . 409
15.7.14 TCustomDaemonApplication.OnRun . . . 409
15.7.15 TCustomDaemonApplication.EventLog . . . 410
15.7.16 TCustomDaemonApplication.GUIMainLoop . . . 410

15.7.17 TCustomDaemonApplication.GuiHandle . . . 410
15.7.18 TCustomDaemonApplication.RunMode . . . 410
15.7.19 TCustomDaemonApplication.AutoRegisterMessageFile . . . 411
15.8 TCustomDaemonMapper . . . 411
15.8.1 Description . . . 411
15.8.2 Method overview . . . 411
15.8.3 Property overview . . . 411
15.8.4 TCustomDaemonMapper.Create . . . 411
15.8.5 TCustomDaemonMapper.Destroy . . . 412
15.8.6 TCustomDaemonMapper.DaemonDefs . . . 412
15.8.7 TCustomDaemonMapper.OnCreate . . . 412
15.8.8 TCustomDaemonMapper.OnDestroy . . . 413
15.8.9 TCustomDaemonMapper.OnRun . . . 413
15.8.10 TCustomDaemonMapper.OnInstall . . . 413
15.8.11 TCustomDaemonMapper.OnUnInstall . . . 413
15.9 TDaemon . . . 414
15.9.1 Description . . . 414
15.9.2 Property overview . . . 414
15.9.3 TDaemon.Definition . . . 414
15.9.4 TDaemon.Status . . . 414
15.9.5 TDaemon.OnStart . . . 414
15.9.6 TDaemon.OnStop . . . 415
15.9.7 TDaemon.OnPause . . . 415
15.9.8 TDaemon.OnContinue . . . 415
15.9.9 TDaemon.OnShutDown . . . 416
15.9.10 TDaemon.OnExecute . . . 416
15.9.11 TDaemon.BeforeInstall . . . 416
15.9.12 TDaemon.AfterInstall . . . 417
15.9.13 TDaemon.BeforeUnInstall . . . 417
15.9.14 TDaemon.AfterUnInstall . . . 417
15.9.15 TDaemon.OnControlCode . . . 417
15.9.16 TDaemon.OnControlCodeEvent . . . 418
15.10 TDaemonApplication . . . 418
15.10.1 Description . . . 418
15.11 TDaemonController . . . 418
15.11.1 Description . . . 418
15.11.2 Method overview . . . 418
15.11.3 Property overview . . . 418
15.11.4 TDaemonController.Create . . . 419
15.11.5 TDaemonController.Destroy . . . 419

15.11.6 TDaemonController.StartService . . . 419
15.11.7 TDaemonController.Main . . . 419
15.11.8 TDaemonController.Controller . . . 420
15.11.9 TDaemonController.ReportStatus . . . 420
15.11.10TDaemonController.Daemon . . . 420
15.11.11TDaemonController.Params . . . 420
15.11.12TDaemonController.LastStatus . . . 421
15.11.13TDaemonController.CheckPoint . . . 421
15.12TDaemonDef . . . 421
15.12.1 Description . . . 421
15.12.2 Method overview . . . 421
15.12.3 Property overview . . . 422
15.12.4 TDaemonDef.Create . . . 422
15.12.5 TDaemonDef.Destroy . . . 422
15.12.6 TDaemonDef.DaemonClass . . . 422
15.12.7 TDaemonDef.Instance . . . 423
15.12.8 TDaemonDef.DaemonClassName . . . 423
15.12.9 TDaemonDef.Name . . . 423
15.12.10TDaemonDef.Description . . . 423
15.12.11TDaemonDef.DisplayName . . . 424
15.12.12TDaemonDef.RunArguments . . . 424
15.12.13TDaemonDef.Options . . . 424
15.12.14TDaemonDef.Enabled . . . 424
15.12.15TDaemonDef.WinBindings . . . 425
15.12.16TDaemonDef.OnCreateInstance . . . 425
15.12.17TDaemonDef.LogStatusReport . . . 425
15.13TDaemonDefs . . . 425
15.13.1 Description . . . 425
15.13.2 Method overview . . . 426
15.13.3 Property overview . . . 426
15.13.4 TDaemonDefs.Create . . . 426
15.13.5 TDaemonDefs.IndexOfDaemonDef . . . 426
15.13.6 TDaemonDefs.FindDaemonDef . . . 426
15.13.7 TDaemonDefs.DaemonDefByName . . . 427
15.13.8 TDaemonDefs.Daemons . . . 427
15.14TDaemonMapper . . . 427
15.14.1 Description . . . 427
15.14.2 Method overview . . . 427
15.14.3 TDaemonMapper.Create . . . 428
15.14.4 TDaemonMapper.CreateNew . . . 428

15.15TDaemonThread 428
15.15.1 Description 428
15.15.2 Method overview 428
15.15.3 Property overview 428
15.15.4 TDaemonThread.Create 429
15.15.5 TDaemonThread.Execute 429
15.15.6 TDaemonThread.CheckControlMessage 429
15.15.7 TDaemonThread.StopDaemon 429
15.15.8 TDaemonThread.PauseDaemon 430
15.15.9 TDaemonThread.ContinueDaemon 430
15.15.10TDaemonThread.ShutDownDaemon 430
15.15.11TDaemonThread.InterrogateDaemon 430
15.15.12TDaemonThread.Daemon 431
15.16TDependencies 431
15.16.1 Description 431
15.16.2 Method overview 431
15.16.3 Property overview 431
15.16.4 TDependencies.Create 431
15.16.5 TDependencies.Items 431
15.17TDependency 432
15.17.1 Description 432
15.17.2 Method overview 432
15.17.3 Property overview 432
15.17.4 TDependency.Assign 432
15.17.5 TDependency.Name 432
15.17.6 TDependency.IsGroup 432
15.18TWinBindings 433
15.18.1 Description 433
15.18.2 Method overview 433
15.18.3 Property overview 433
15.18.4 TWinBindings.Create 433
15.18.5 TWinBindings.Destroy 433
15.18.6 TWinBindings.Assign 434
15.18.7 TWinBindings.ErrCode 434
15.18.8 TWinBindings.Win32ErrCode 434
15.18.9 TWinBindings.Dependencies 434
15.18.10TWinBindings.GroupName 435
15.18.11TWinBindings.Password 435
15.18.12TWinBindings.UserName 435
15.18.13TWinBindings.StartType 435

15.18.14TWinBindings.WaitHint . . . 436
15.18.15TWinBindings.IDTag . . . 436
15.18.16TWinBindings.ServiceType . . . 436
15.18.17TWinBindings.ErrorSeverity . . . 436
15.18.18TWinBindings.AcceptedCodes . . . 437

16 Reference for unit 'DB' 438
16.1 Used units . . . 438
16.2 Overview . . . 438
16.3 Constants, types and variables . . . 438
16.3.1 Constants . . . 438
16.3.2 Types . . . 440
16.3.3 Variables . . . 455
16.4 Procedures and functions . . . 456
16.4.1 BuffersEqual . . . 456
16.4.2 DatabaseError . . . 456
16.4.3 DatabaseErrorFmt . . . 456
16.4.4 DateTimeRecToDateTime . . . 457
16.4.5 DateTimeToDateTimeRec . . . 457
16.4.6 DisposeMem . . . 457
16.4.7 enumerator(TDataSet):TDataSetEnumerator . . . 457
16.4.8 ExtractFieldName . . . 458
16.4.9 SkipComments . . . 458
16.5 TLookupListRec . . . 458
16.6 EDatabaseError . . . 458
16.6.1 Description . . . 458
16.7 EUpdateError . . . 459
16.7.1 Description . . . 459
16.7.2 Method overview . . . 459
16.7.3 Property overview . . . 459
16.7.4 EUpdateError.Create . . . 459
16.7.5 EUpdateError.Destroy . . . 459
16.7.6 EUpdateError.Context . . . 460
16.7.7 EUpdateError.ErrorCode . . . 460
16.7.8 EUpdateError.OriginalException . . . 460
16.7.9 EUpdateError.PreviousError . . . 460
16.8 IProviderSupport . . . 461
16.8.1 Description . . . 461
16.8.2 Method overview . . . 461
16.8.3 IProviderSupport.PSEndTransaction . . . 461

16.8.4 IProviderSupport.PSExecute . . . 461
16.8.5 IProviderSupport.PSExecuteStatement . . . 462
16.8.6 IProviderSupport.PSGetAttributes . . . 462
16.8.7 IProviderSupport.PSGetCommandText . . . 462
16.8.8 IProviderSupport.PSGetCommandType . . . 463
16.8.9 IProviderSupport.PSGetDefaultOrder . . . 463
16.8.10 IProviderSupport.PSGetIndexDefs . . . 463
16.8.11 IProviderSupport.PSGetKeyFields . . . 463
16.8.12 IProviderSupport.PSGetParams . . . 464
16.8.13 IProviderSupport.PSGetQuoteChar . . . 464
16.8.14 IProviderSupport.PSGetTableName . . . 464
16.8.15 IProviderSupport.PSGetUpdateException . . . 464
16.8.16 IProviderSupport.PSInTransaction . . . 465
16.8.17 IProviderSupport.PSIsSQLBased . . . 465
16.8.18 IProviderSupport.PSIsSQLSupported . . . 465
16.8.19 IProviderSupport.PSReset . . . 465
16.8.20 IProviderSupport.PSSetCommandText . . . 466
16.8.21 IProviderSupport.PSSetParams . . . 466
16.8.22 IProviderSupport.PSStartTransaction . . . 466
16.8.23 IProviderSupport.PSUpdateRecord . . . 466
16.9 TArrayField . . . 467
16.9.1 Method overview . . . 467
16.9.2 TArrayField.Create . . . 467
16.10 TAutoIncField . . . 467
16.10.1 Description . . . 467
16.10.2 Method overview . . . 467
16.10.3 TAutoIncField.Create . . . 467
16.11 TBCDField . . . 467
16.11.1 Description . . . 467
16.11.2 Method overview . . . 468
16.11.3 Property overview . . . 468
16.11.4 TBCDField.Create . . . 468
16.11.5 TBCDField.CheckRange . . . 468
16.11.6 TBCDField.Value . . . 469
16.11.7 TBCDField.Precision . . . 469
16.11.8 TBCDField.Currency . . . 469
16.11.9 TBCDField.MaxValue . . . 469
16.11.10 TBCDField.MinValue . . . 470
16.11.11 TBCDField.Size . . . 470
16.12 TBinaryField . . . 470

16.12.1 Description . . . 470
16.12.2 Method overview . . . 471
16.12.3 Property overview . . . 471
16.12.4 TBinaryField.Create . . . 471
16.12.5 TBinaryField.Size . . . 471
16.13TBlobField . . . 471
16.13.1 Description . . . 471
16.13.2 Method overview . . . 472
16.13.3 Property overview . . . 472
16.13.4 TBlobField.Create . . . 472
16.13.5 TBlobField.Clear . . . 472
16.13.6 TBlobField.IsBlob . . . 472
16.13.7 TBlobField.LoadFromFile . . . 473
16.13.8 TBlobField.LoadFromStream . . . 473
16.13.9 TBlobField.SaveToFile . . . 473
16.13.10TBlobField.SaveToStream . . . 474
16.13.11TBlobField.SetFieldType . . . 474
16.13.12TBlobField.BlobSize . . . 474
16.13.13TBlobField.Modified . . . 474
16.13.14TBlobField.Value . . . 475
16.13.15TBlobField.Transliterate . . . 475
16.13.16TBlobField.BlobType . . . 475
16.13.17TBlobField.Size . . . 475
16.14TBooleanField . . . 476
16.14.1 Description . . . 476
16.14.2 Method overview . . . 476
16.14.3 Property overview . . . 476
16.14.4 TBooleanField.Create . . . 476
16.14.5 TBooleanField.Value . . . 476
16.14.6 TBooleanField.DisplayValues . . . 477
16.15TByteField . . . 477
16.15.1 Description . . . 477
16.15.2 Method overview . . . 477
16.15.3 TByteField.Create . . . 477
16.16TBytesField . . . 478
16.16.1 Description . . . 478
16.16.2 Method overview . . . 478
16.16.3 TBytesField.Create . . . 478
16.17TCheckConstraint . . . 478
16.17.1 Description . . . 478

16.17.2 Method overview . . . 478
16.17.3 Property overview . . . 479
16.17.4 TCheckConstraint.Assign . . . 479
16.17.5 TCheckConstraint.CustomConstraint . . . 479
16.17.6 TCheckConstraint.ErrorMessage . . . 479
16.17.7 TCheckConstraint.FromDictionary . . . 480
16.17.8 TCheckConstraint.ImportedConstraint . . . 480
16.18TCheckConstraints . . . 480
16.18.1 Description . . . 480
16.18.2 Method overview . . . 480
16.18.3 Property overview . . . 480
16.18.4 TCheckConstraints.Create . . . 481
16.18.5 TCheckConstraints.Add . . . 481
16.18.6 TCheckConstraints.Items . . . 481
16.19TCurrencyField . . . 481
16.19.1 Description . . . 481
16.19.2 Method overview . . . 482
16.19.3 Property overview . . . 482
16.19.4 TCurrencyField.Create . . . 482
16.19.5 TCurrencyField.Currency . . . 482
16.20TCustomConnection . . . 482
16.20.1 Description . . . 482
16.20.2 Method overview . . . 482
16.20.3 Property overview . . . 483
16.20.4 TCustomConnection.Close . . . 483
16.20.5 TCustomConnection.Destroy . . . 483
16.20.6 TCustomConnection.Open . . . 483
16.20.7 TCustomConnection.DataSetCount . . . 484
16.20.8 TCustomConnection.DataSets . . . 484
16.20.9 TCustomConnection.Connected . . . 484
16.20.10TCustomConnection.LoginPrompt . . . 485
16.20.11TCustomConnection.AfterConnect . . . 485
16.20.12TCustomConnection.AfterDisconnect . . . 485
16.20.13TCustomConnection.BeforeConnect . . . 486
16.20.14TCustomConnection.BeforeDisconnect . . . 486
16.20.15TCustomConnection.OnLogin . . . 486
16.20.16TCustomConnection.OnCloseError . . . 486
16.21TDatabase . . . 487
16.21.1 Description . . . 487
16.21.2 Method overview . . . 487

16.21.3 Property overview . . . 487
16.21.4 TDatabase.Create . . . 487
16.21.5 TDatabase.Destroy . . . 488
16.21.6 TDatabase.CloseDataSets . . . 488
16.21.7 TDatabase.CloseTransactions . . . 488
16.21.8 TDatabase.StartTransaction . . . 488
16.21.9 TDatabase.EndTransaction . . . 489
16.21.10TDatabase.TransactionCount . . . 489
16.21.11TDatabase.Transactions . . . 489
16.21.12TDatabase.Directory . . . 489
16.21.13TDatabase.IsSQLBased . . . 490
16.21.14TDatabase.Connected . . . 490
16.21.15TDatabase.DatabaseName . . . 490
16.21.16TDatabase.KeepConnection . . . 490
16.21.17TDatabase.Params . . . 491
16.22TDataLink . . . 491
16.22.1 Description . . . 491
16.22.2 Method overview . . . 491
16.22.3 Property overview . . . 492
16.22.4 TDataLink.Create . . . 492
16.22.5 TDataLink.Destroy . . . 492
16.22.6 TDataLink.Edit . . . 492
16.22.7 TDataLink.UpdateRecord . . . 493
16.22.8 TDataLink.ExecuteAction . . . 493
16.22.9 TDataLink.UpdateAction . . . 493
16.22.10TDataLink.Active . . . 493
16.22.11TDataLink.ActiveRecord . . . 494
16.22.12TDataLink.BOF . . . 494
16.22.13TDataLink.BufferCount . . . 494
16.22.14TDataLink.DataSet . . . 495
16.22.15TDataLink.DataSource . . . 495
16.22.16TDataLink.DataSourceFixed . . . 495
16.22.17TDataLink.Editing . . . 495
16.22.18TDataLink.Eof . . . 496
16.22.19TDataLink.ReadOnly . . . 496
16.22.20TDataLink.RecordCount . . . 496
16.23TDataSet . . . 496
16.23.1 Description . . . 496
16.23.2 Method overview . . . 499
16.23.3 Property overview . . . 501

16.23.4 TDataSet.Create . . . 502
16.23.5 TDataSet.Destroy . . . 502
16.23.6 TDataSet.ActiveBuffer . . . 502
16.23.7 TDataSet.GetFieldData . . . 502
16.23.8 TDataSet.SetFieldData . . . 503
16.23.9 TDataSet.Append . . . 503
16.23.10 TDataSet.AppendRecord . . . 503
16.23.11 TDataSet.BookmarkValid . . . 504
16.23.12 TDataSet.Cancel . . . 504
16.23.13 TDataSet.CheckBrowseMode . . . 504
16.23.14 TDataSet.ClearFields . . . 504
16.23.15 TDataSet.Close . . . 505
16.23.16 TDataSet.ControlsDisabled . . . 505
16.23.17 TDataSet.CompareBookmarks . . . 505
16.23.18 TDataSet.CreateBlobStream . . . 506
16.23.19 TDataSet.CursorPosChanged . . . 506
16.23.20 TDataSet.DataConvert . . . 506
16.23.21 TDataSet.Delete . . . 506
16.23.22 TDataSet.DisableControls . . . 507
16.23.23 TDataSet.Edit . . . 507
16.23.24 TDataSet.EnableControls . . . 508
16.23.25 TDataSet.FieldByName . . . 508
16.23.26 TDataSet.FindField . . . 508
16.23.27 TDataSet.FindFirst . . . 509
16.23.28 TDataSet.FindLast . . . 509
16.23.29 TDataSet.FindNext . . . 509
16.23.30 TDataSet.FindPrior . . . 509
16.23.31 TDataSet.First . . . 510
16.23.32 TDataSet.FreeBookmark . . . 510
16.23.33 TDataSet.GetBookmark . . . 510
16.23.34 TDataSet.GetCurrentRecord . . . 511
16.23.35 TDataSet.GetFieldList . . . 511
16.23.36 TDataSet.GetFieldNames . . . 511
16.23.37 TDataSet.GotoBookmark . . . 511
16.23.38 TDataSet.Insert . . . 512
16.23.39 TDataSet.InsertRecord . . . 512
16.23.40 TDataSet.IsEmpty . . . 512
16.23.41 TDataSet.IsLinkedTo . . . 512
16.23.42 TDataSet.IsSequenced . . . 513
16.23.43 TDataSet.Last . . . 513

16.23.44TDataSet.Locate . . . 513
16.23.45TDataSet.Lookup . . . 514
16.23.46TDataSet.MoveBy . . . 514
16.23.47TDataSet.Next . . . 514
16.23.48TDataSet.Open . . . 515
16.23.49TDataSet.Post . . . 515
16.23.50TDataSet.Prior . . . 516
16.23.51TDataSet.Refresh . . . 516
16.23.52TDataSet.Resync . . . 516
16.23.53TDataSet.SetFields . . . 516
16.23.54TDataSet.Translate . . . 517
16.23.55TDataSet.UpdateCursorPos . . . 517
16.23.56TDataSet.UpdateRecord . . . 517
16.23.57TDataSet.UpdateStatus . . . 518
16.23.58TDataSet.BlockReadSize . . . 518
16.23.59TDataSet.BOF . . . 518
16.23.60TDataSet.Bookmark . . . 518
16.23.61TDataSet.CanModify . . . 519
16.23.62TDataSet.DataSource . . . 520
16.23.63TDataSet.DefaultFields . . . 520
16.23.64TDataSet.EOF . . . 520
16.23.65TDataSet.FieldCount . . . 521
16.23.66TDataSet.FieldDefs . . . 521
16.23.67TDataSet.Found . . . 522
16.23.68TDataSet.Modified . . . 522
16.23.69TDataSet.IsUniDirectional . . . 522
16.23.70TDataSet.RecordCount . . . 523
16.23.71TDataSet.RecNo . . . 523
16.23.72TDataSet.RecordSize . . . 523
16.23.73TDataSet.SparseArrays . . . 524
16.23.74TDataSet.State . . . 524
16.23.75TDataSet.Fields . . . 524
16.23.76TDataSet.FieldValues . . . 525
16.23.77TDataSet.Filter . . . 525
16.23.78TDataSet.Filtered . . . 525
16.23.79TDataSet.FilterOptions . . . 526
16.23.80TDataSet.Active . . . 526
16.23.81TDataSet.AutoCalcFields . . . 526
16.23.82TDataSet.BeforeOpen . . . 527
16.23.83TDataSet.AfterOpen . . . 527

16.23.84TDataSet.BeforeClose . . . 527
16.23.85TDataSet.AfterClose . . . 527
16.23.86TDataSet.BeforeInsert . . . 528
16.23.87TDataSet.AfterInsert . . . 528
16.23.88TDataSet.BeforeEdit . . . 528
16.23.89TDataSet.AfterEdit . . . 529
16.23.90TDataSet.BeforePost . . . 529
16.23.91TDataSet.AfterPost . . . 529
16.23.92TDataSet.BeforeCancel . . . 530
16.23.93TDataSet.AfterCancel . . . 530
16.23.94TDataSet.BeforeDelete . . . 530
16.23.95TDataSet.AfterDelete . . . 531
16.23.96TDataSet.BeforeScroll . . . 531
16.23.97TDataSet.AfterScroll . . . 531
16.23.98TDataSet.BeforeRefresh . . . 532
16.23.99TDataSet.AfterRefresh . . . 532
16.23.100DataSet.OnCalcFields . . . 532
16.23.101DataSet.OnDeleteError . . . 533
16.23.102DataSet.OnEditError . . . 533
16.23.103DataSet.OnFilterRecord . . . 534
16.23.104DataSet.OnNewRecord . . . 534
16.23.105DataSet.OnPostError . . . 534
16.24TDataSetEnumerator . . . 535
16.24.1 Description . . . 535
16.24.2 Method overview . . . 535
16.24.3 Property overview . . . 535
16.24.4 TDataSetEnumerator.Create . . . 535
16.24.5 TDataSetEnumerator.MoveNext . . . 535
16.24.6 TDataSetEnumerator.Current . . . 536
16.25TDataSource . . . 536
16.25.1 Description . . . 536
16.25.2 Method overview . . . 536
16.25.3 Property overview . . . 536
16.25.4 TDataSource.Create . . . 537
16.25.5 TDataSource.Destroy . . . 537
16.25.6 TDataSource.Edit . . . 537
16.25.7 TDataSource.IsLinkedTo . . . 537
16.25.8 TDataSource.State . . . 538
16.25.9 TDataSource.AutoEdit . . . 538
16.25.10TDataSource.DataSet . . . 538

16.25.1 TDataSource.Enabled . . . 539
16.25.2 TDataSource.OnStateChange . . . 539
16.25.3 TDataSource.OnDataChange . . . 539
16.25.4 TDataSource.OnUpdateData . . . 540
16.26 TDateField . . . 540
16.26.1 Description . . . 540
16.26.2 Method overview . . . 540
16.26.3 TDateField.Create . . . 540
16.27 TDateTimeField . . . 540
16.27.1 Description . . . 540
16.27.2 Method overview . . . 541
16.27.3 Property overview . . . 541
16.27.4 TDateTimeField.Create . . . 541
16.27.5 TDateTimeField.Value . . . 541
16.27.6 TDateTimeField.DisplayFormat . . . 541
16.27.7 TDateTimeField.EditMask . . . 542
16.28 TDBDataset . . . 542
16.28.1 Description . . . 542
16.28.2 Method overview . . . 542
16.28.3 Property overview . . . 542
16.28.4 TDBDataset.destroy . . . 543
16.28.5 TDBDataset.DataBase . . . 543
16.28.6 TDBDataset.Transaction . . . 543
16.29 TDBTransaction . . . 543
16.29.1 Description . . . 543
16.29.2 Method overview . . . 544
16.29.3 Property overview . . . 544
16.29.4 TDBTransaction.Create . . . 544
16.29.5 TDBTransaction.Destroy . . . 544
16.29.6 TDBTransaction.CloseDataSets . . . 544
16.29.7 TDBTransaction.DataBase . . . 545
16.29.8 TDBTransaction.Active . . . 545
16.30 TDefCollection . . . 545
16.30.1 Description . . . 545
16.30.2 Method overview . . . 545
16.30.3 Property overview . . . 545
16.30.4 TDefCollection.create . . . 546
16.30.5 TDefCollection.Find . . . 546
16.30.6 TDefCollection.GetItemNames . . . 546
16.30.7 TDefCollection.IndexOf . . . 546

16.30.8 TDefCollection.Dataset . . . 547
16.30.9 TDefCollection.Updated . . . 547
16.31 TDetailDataLink . . . 547
16.31.1 Description . . . 547
16.31.2 Property overview . . . 547
16.31.3 TDetailDataLink.DetailDataSet . . . 547
16.32 TExtendedField . . . 548
16.32.1 Method overview . . . 548
16.32.2 Property overview . . . 548
16.32.3 TExtendedField.Create . . . 548
16.32.4 TExtendedField.CheckRange . . . 548
16.32.5 TExtendedField.Value . . . 548
16.32.6 TExtendedField.Currency . . . 548
16.32.7 TExtendedField.MaxValue . . . 548
16.32.8 TExtendedField.MinValue . . . 549
16.32.9 TExtendedField.Precision . . . 549
16.33 TField . . . 549
16.33.1 Description . . . 549
16.33.2 Method overview . . . 549
16.33.3 Property overview . . . 551
16.33.4 TField.Create . . . 552
16.33.5 TField.Destroy . . . 552
16.33.6 TField.Assign . . . 552
16.33.7 TField.AssignValue . . . 552
16.33.8 TField.Clear . . . 553
16.33.9 TField.FocusControl . . . 553
16.33.10 TField.GetData . . . 553
16.33.11 TField.IsBlob . . . 554
16.33.12 TField.IsValidChar . . . 554
16.33.13 TField.RefreshLookupList . . . 554
16.33.14 TField.SetData . . . 554
16.33.15 TField.SetFieldType . . . 555
16.33.16 TField.Validate . . . 555
16.33.17 TField.AsBCD . . . 555
16.33.18 TField.AsBoolean . . . 556
16.33.19 TField.AsBytes . . . 556
16.33.20 TField.AsCurrency . . . 556
16.33.21 TField.AsDateTime . . . 557
16.33.22 TField.AsExtended . . . 557
16.33.23 TField.AsFloat . . . 557

16.33.24 TField.AsLongint . . . 557
16.33.25 TField.AsLongWord . . . 558
16.33.26 TField.AsLargeInt . . . 558
16.33.27 TField.AsInteger . . . 558
16.33.28 TField.AsSingle . . . 559
16.33.29 TField.AsString . . . 559
16.33.30 TField.AsAnsiString . . . 559
16.33.31 TField.AsUnicodeString . . . 560
16.33.32 TField.AsUTF8String . . . 560
16.33.33 TField.AsWideString . . . 560
16.33.34 TField.AsVariant . . . 561
16.33.35 TField.AttributeSet . . . 561
16.33.36 TField.Calculated . . . 561
16.33.37 TField.CanModify . . . 561
16.33.38 TField.CurValue . . . 562
16.33.39 TField.DataSet . . . 562
16.33.40 TField.DataSize . . . 562
16.33.41 TField.DataType . . . 562
16.33.42 TField.DisplayName . . . 563
16.33.43 TField.DisplayText . . . 563
16.33.44 TField.EditMask . . . 563
16.33.45 TField.EditMaskPtr . . . 564
16.33.46 TField.FieldNo . . . 564
16.33.47 TField.IsIndexField . . . 564
16.33.48 TField.IsNull . . . 564
16.33.49 TField.NewValue . . . 565
16.33.50 TField.Offset . . . 565
16.33.51 TField.Size . . . 565
16.33.52 TField.Text . . . 565
16.33.53 TField.ValidChars . . . 566
16.33.54 TField.Value . . . 566
16.33.55 TField.OldValue . . . 566
16.33.56 TField.LookupList . . . 567
16.33.57 TField.FieldDef . . . 567
16.33.58 TField.Alignment . . . 567
16.33.59 TField.CustomConstraint . . . 568
16.33.60 TField.ConstraintErrorMessage . . . 568
16.33.61 TField.DefaultExpression . . . 568
16.33.62 TField.DisplayLabel . . . 569
16.33.63 TField.DisplayWidth . . . 569

16.33.64TField.FieldKind . . . 569
16.33.65TField.FieldName . . . 569
16.33.66TField.HasConstraints . . . 570
16.33.67TField.Index . . . 570
16.33.68TField.ImportedConstraint . . . 570
16.33.69TField.KeyFields . . . 571
16.33.70TField.LookupCache . . . 571
16.33.71TField.LookupDataSet . . . 571
16.33.72TField.LookupKeyFields . . . 571
16.33.73TField.LookupResultField . . . 572
16.33.74TField.Lookup . . . 572
16.33.75TField.Origin . . . 572
16.33.76TField.ParentField . . . 572
16.33.77TField.ProviderFlags . . . 573
16.33.78TField.ReadOnly . . . 573
16.33.79TField.Required . . . 573
16.33.80TField.Visible . . . 574
16.33.81TField.OnChange . . . 574
16.33.82TField.OnGetText . . . 574
16.33.83TField.OnSetText . . . 574
16.33.84TField.OnValidate . . . 575
16.34TFieldDef . . . 575
16.34.1 Description . . . 575
16.34.2 Method overview . . . 575
16.34.3 Property overview . . . 576
16.34.4 TFieldDef.Create . . . 576
16.34.5 TFieldDef.Destroy . . . 576
16.34.6 TFieldDef.AddChild . . . 577
16.34.7 TFieldDef.Assign . . . 577
16.34.8 TFieldDef.CreateField . . . 577
16.34.9 TFieldDef.HasChildDefs . . . 577
16.34.10TFieldDef.FieldClass . . . 577
16.34.11TFieldDef.FieldNo . . . 578
16.34.12TFieldDef.CharSize . . . 578
16.34.13TFieldDef.InternalCalcField . . . 578
16.34.14TFieldDef.ParentDef . . . 579
16.34.15TFieldDef.Required . . . 579
16.34.16TFieldDef.Codepage . . . 579
16.34.17TFieldDef.Attributes . . . 579
16.34.18TFieldDef.DataType . . . 580

16.34.19TFieldDef.ChildDefs . . . 580
16.34.20TFieldDef.Precision . . . 580
16.34.21TFieldDef.Size . . . 580
16.35TFieldDefs . . . 581
16.35.1 Description . . . 581
16.35.2 Method overview . . . 581
16.35.3 Property overview . . . 581
16.35.4 TFieldDefs.Create . . . 581
16.35.5 TFieldDefs.Add . . . 581
16.35.6 TFieldDefs.AddFieldDef . . . 582
16.35.7 TFieldDefs.Assign . . . 582
16.35.8 TFieldDefs.Find . . . 582
16.35.9 TFieldDefs.Update . . . 583
16.35.10TFieldDefs.MakeNameUnique . . . 583
16.35.11TFieldDefs.HiddenFields . . . 583
16.35.12TFieldDefs.Items . . . 583
16.35.13TFieldDefs.ParentDef . . . 584
16.36TFields . . . 584
16.36.1 Description . . . 584
16.36.2 Method overview . . . 584
16.36.3 Property overview . . . 584
16.36.4 TFields.Create . . . 584
16.36.5 TFields.Destroy . . . 585
16.36.6 TFields.Add . . . 585
16.36.7 TFields.CheckFieldName . . . 585
16.36.8 TFields.CheckFieldNames . . . 585
16.36.9 TFields.Clear . . . 586
16.36.10TFields.FindField . . . 586
16.36.11TFields.FieldByName . . . 586
16.36.12TFields.FieldByNumber . . . 586
16.36.13TFields.GetEnumerator . . . 587
16.36.14TFields.GetFieldNames . . . 587
16.36.15TFields.IndexOf . . . 587
16.36.16TFields.Remove . . . 587
16.36.17TFields.Count . . . 588
16.36.18TFields.Dataset . . . 588
16.36.19TFields.Fields . . . 588
16.37TFieldsEnumerator . . . 588
16.37.1 Description . . . 588
16.37.2 Method overview . . . 589

16.37.3 Property overview . . . 589
16.37.4 TFieldsEnumerator.Create . . . 589
16.37.5 TFieldsEnumerator.MoveNext . . . 589
16.37.6 TFieldsEnumerator.Current . . . 590
16.38 TFloatField . . . 590
16.38.1 Description . . . 590
16.38.2 Method overview . . . 590
16.38.3 Property overview . . . 590
16.38.4 TFloatField.Create . . . 590
16.38.5 TFloatField.CheckRange . . . 591
16.38.6 TFloatField.Value . . . 591
16.38.7 TFloatField.Currency . . . 591
16.38.8 TFloatField.MaxValue . . . 591
16.38.9 TFloatField.MinValue . . . 592
16.38.10 TFloatField.Precision . . . 592
16.39 TFMTBCDField . . . 592
16.39.1 Description . . . 592
16.39.2 Method overview . . . 593
16.39.3 Property overview . . . 593
16.39.4 TFMTBCDField.Create . . . 593
16.39.5 TFMTBCDField.CheckRange . . . 593
16.39.6 TFMTBCDField.Value . . . 593
16.39.7 TFMTBCDField.Precision . . . 594
16.39.8 TFMTBCDField.Currency . . . 594
16.39.9 TFMTBCDField.MaxValue . . . 594
16.39.10 TFMTBCDField.MinValue . . . 594
16.39.11 TFMTBCDField.Size . . . 595
16.40 TGraphicField . . . 595
16.40.1 Description . . . 595
16.40.2 Method overview . . . 595
16.40.3 TGraphicField.Create . . . 595
16.41 TGuidField . . . 595
16.41.1 Description . . . 595
16.41.2 Method overview . . . 596
16.41.3 Property overview . . . 596
16.41.4 TGuidField.Create . . . 596
16.41.5 TGuidField.AsGuid . . . 596
16.42 TIndexDef . . . 596
16.42.1 Description . . . 596
16.42.2 Method overview . . . 597

16.42.3 Property overview . . . 597
16.42.4 TIndexDef.Create . . . 597
16.42.5 TIndexDef.Expression . . . 597
16.42.6 TIndexDef.Fields . . . 597
16.42.7 TIndexDef.CaseInsFields . . . 598
16.42.8 TIndexDef.DescFields . . . 598
16.42.9 TIndexDef.Options . . . 598
16.42.10TIndexDef.Source . . . 599
16.43TIndexDefs . . . 599
16.43.1 Description . . . 599
16.43.2 Method overview . . . 599
16.43.3 Property overview . . . 599
16.43.4 TIndexDefs.Create . . . 599
16.43.5 TIndexDefs.Add . . . 600
16.43.6 TIndexDefs.AddIndexDef . . . 600
16.43.7 TIndexDefs.Find . . . 600
16.43.8 TIndexDefs.FindIndexForFields . . . 600
16.43.9 TIndexDefs.GetIndexForFields . . . 601
16.43.10TIndexDefs.Update . . . 601
16.43.11TIndexDefs.Items . . . 601
16.44TIntegerField . . . 601
16.44.1 Description . . . 601
16.45TLargeintField . . . 601
16.45.1 Description . . . 601
16.45.2 Method overview . . . 602
16.45.3 Property overview . . . 602
16.45.4 TLargeintField.Create . . . 602
16.45.5 TLargeintField.CheckRange . . . 602
16.45.6 TLargeintField.Value . . . 602
16.45.7 TLargeintField.MaxValue . . . 603
16.45.8 TLargeintField.MinValue . . . 603
16.46TLongintField . . . 603
16.46.1 Description . . . 603
16.46.2 Method overview . . . 604
16.46.3 Property overview . . . 604
16.46.4 TLongintField.Create . . . 604
16.46.5 TLongintField.CheckRange . . . 604
16.46.6 TLongintField.Value . . . 604
16.46.7 TLongintField.MaxValue . . . 605
16.46.8 TLongintField.MinValue . . . 605

16.47TLongWordField . . . 605
16.47.1 Description . . . 605
16.47.2 Method overview . . . 605
16.47.3 Property overview . . . 605
16.47.4 TLongWordField.Create . . . 606
16.47.5 TLongWordField.CheckRange . . . 606
16.47.6 TLongWordField.Value . . . 606
16.47.7 TLongWordField.MaxValue . . . 606
16.47.8 TLongWordField.MinValue . . . 607
16.48TLookupList . . . 607
16.48.1 Description . . . 607
16.48.2 Method overview . . . 607
16.48.3 TLookupList.Create . . . 607
16.48.4 TLookupList.Destroy . . . 607
16.48.5 TLookupList.Add . . . 608
16.48.6 TLookupList.Clear . . . 608
16.48.7 TLookupList.FirstKeyByValue . . . 608
16.48.8 TLookupList.ValueOfKey . . . 608
16.48.9 TLookupList.ValuesToStrings . . . 609
16.49TMasterDataLink . . . 609
16.49.1 Description . . . 609
16.49.2 Method overview . . . 609
16.49.3 Property overview . . . 609
16.49.4 TMasterDataLink.Create . . . 609
16.49.5 TMasterDataLink.Destroy . . . 610
16.49.6 TMasterDataLink.FieldNames . . . 610
16.49.7 TMasterDataLink.Fields . . . 610
16.49.8 TMasterDataLink.OnMasterChange . . . 610
16.49.9 TMasterDataLink.OnMasterDisable . . . 611
16.50TMasterParamsDataLink . . . 611
16.50.1 Description . . . 611
16.50.2 Method overview . . . 611
16.50.3 Property overview . . . 611
16.50.4 TMasterParamsDataLink.Create . . . 611
16.50.5 TMasterParamsDataLink.RefreshParamNames . . . 612
16.50.6 TMasterParamsDataLink.CopyParamsFromMaster . . . 612
16.50.7 TMasterParamsDataLink.Params . . . 612
16.51TMemoField . . . 612
16.51.1 Description . . . 612
16.51.2 Method overview . . . 613

16.51.3 Property overview . . . 613
16.51.4 TMemoField.Create . . . 613
16.51.5 TMemoField.CodePage . . . 613
16.51.6 TMemoField.Transliterate . . . 613
16.52TNamedItem . . . 614
16.52.1 Description . . . 614
16.52.2 Property overview . . . 614
16.52.3 TNamedItem.DisplayName . . . 614
16.52.4 TNamedItem.Name . . . 614
16.53TNumericField . . . 614
16.53.1 Description . . . 614
16.53.2 Method overview . . . 615
16.53.3 Property overview . . . 615
16.53.4 TNumericField.Create . . . 615
16.53.5 TNumericField.Alignment . . . 615
16.53.6 TNumericField.DisplayFormat . . . 615
16.53.7 TNumericField.EditFormat . . . 616
16.54TObjectField . . . 616
16.54.1 Property overview . . . 616
16.54.2 TObjectField.FieldCount . . . 616
16.54.3 TObjectField.Fields . . . 616
16.54.4 TObjectField.FieldValues . . . 616
16.54.5 TObjectField.UnNamed . . . 617
16.54.6 TObjectField.ObjectType . . . 617
16.55TParam . . . 617
16.55.1 Description . . . 617
16.55.2 Method overview . . . 617
16.55.3 Property overview . . . 618
16.55.4 TParam.Create . . . 618
16.55.5 TParam.Assign . . . 619
16.55.6 TParam.AssignField . . . 619
16.55.7 TParam.AssignToField . . . 619
16.55.8 TParam.AssignFieldValue . . . 619
16.55.9 TParam.AssignFromField . . . 620
16.55.10TParam.Clear . . . 620
16.55.11TParam.GetData . . . 620
16.55.12TParam.GetDataSize . . . 620
16.55.13TParam.LoadFromFile . . . 621
16.55.14TParam.LoadFromStream . . . 621
16.55.15TParam.SetBlobData . . . 621

16.55.16TParam.SetData 621
16.55.17TParam.AsBCD 622
16.55.18TParam.AsBlob 622
16.55.19TParam.AsBoolean 622
16.55.20TParam.AsByte 623
16.55.21TParam.AsBytes 623
16.55.22TParam.AsCurrency 623
16.55.23TParam.AsDate 623
16.55.24TParam.AsDateTime 624
16.55.25TParam.AsFloat 624
16.55.26TParam.AsInteger 624
16.55.27TParam.AsLargeInt 624
16.55.28TParam.AsLongWord 625
16.55.29TParam.AsMemo 625
16.55.30TParam.AsShortInt 625
16.55.31TParam.AsSingle 625
16.55.32TParam.AsSmallInt 625
16.55.33TParam.AsString 626
16.55.34TParam.AsAnsiString 626
16.55.35TParam.AsUTF8String 626
16.55.36TParam.AsUnicodeString 627
16.55.37TParam.AsTime 627
16.55.38TParam.AsWord 627
16.55.39TParam.AsFMTBCD 627
16.55.40TParam.Bound 628
16.55.41TParam.Dataset 628
16.55.42TParam.IsNull 628
16.55.43TParam.NativeStr 628
16.55.44TParam.Text 629
16.55.45TParam.AsWideString 629
16.55.46TParam.DataType 629
16.55.47TParam.Name 629
16.55.48TParam.NumericScale 630
16.55.49TParam.ParamType 630
16.55.50TParam.Precision 630
16.55.51TParam.Size 631
16.55.52TParam.Value 631
16.56TParams 631
16.56.1 Description 631
16.56.2 Method overview 632

16.56.3 Property overview . . . 632
16.56.4 TParams.Create . . . 632
16.56.5 TParams.AddParam . . . 632
16.56.6 TParams.AssignValues . . . 633
16.56.7 TParams.CreateParam . . . 633
16.56.8 TParams.FindParam . . . 633
16.56.9 TParams.GetParamList . . . 633
16.56.10TParams.IsEqual . . . 634
16.56.11TParams.GetEnumerator . . . 634
16.56.12TParams.ParamByName . . . 634
16.56.13TParams.ParseSQL . . . 634
16.56.14TParams.RemoveParam . . . 635
16.56.15TParams.CopyParamValuesFromDataset . . . 636
16.56.16TParams.Dataset . . . 636
16.56.17TParams.Items . . . 636
16.56.18TParams.ParamValues . . . 636
16.57TParamsEnumerator . . . 637
16.57.1 Description . . . 637
16.57.2 Method overview . . . 637
16.57.3 Property overview . . . 637
16.57.4 TParamsEnumerator.Create . . . 637
16.57.5 TParamsEnumerator.MoveNext . . . 637
16.57.6 TParamsEnumerator.Current . . . 638
16.58TShortintField . . . 638
16.58.1 Description . . . 638
16.58.2 Method overview . . . 638
16.58.3 TShortintField.Create . . . 638
16.59TSingleField . . . 638
16.59.1 Method overview . . . 638
16.59.2 Property overview . . . 639
16.59.3 TSingleField.Create . . . 639
16.59.4 TSingleField.CheckRange . . . 639
16.59.5 TSingleField.Value . . . 639
16.59.6 TSingleField.Currency . . . 639
16.59.7 TSingleField.MaxValue . . . 639
16.59.8 TSingleField.MinValue . . . 639
16.59.9 TSingleField.Precision . . . 640
16.60TSmallintField . . . 640
16.60.1 Description . . . 640
16.60.2 Method overview . . . 640

16.60.3 TSmallintField.Create . . . 640
16.61 TStringField . . . 640
16.61.1 Description . . . 640
16.61.2 Method overview . . . 641
16.61.3 Property overview . . . 641
16.61.4 TStringField.Create . . . 641
16.61.5 TStringField.SetFieldType . . . 641
16.61.6 TStringField.CodePage . . . 641
16.61.7 TStringField.FixedChar . . . 642
16.61.8 TStringField.Transliterate . . . 642
16.61.9 TStringField.Value . . . 642
16.61.10 TStringField.EditMask . . . 642
16.61.11 TStringField.Size . . . 643
16.62 TTimeField . . . 643
16.62.1 Description . . . 643
16.62.2 Method overview . . . 643
16.62.3 TTimeField.Create . . . 643
16.63 TVarBytesField . . . 644
16.63.1 Description . . . 644
16.63.2 Method overview . . . 644
16.63.3 TVarBytesField.Create . . . 644
16.64 TVariantField . . . 644
16.64.1 Description . . . 644
16.64.2 Method overview . . . 644
16.64.3 TVariantField.Create . . . 645
16.65 TWideMemoField . . . 645
16.65.1 Description . . . 645
16.65.2 Method overview . . . 645
16.65.3 Property overview . . . 645
16.65.4 TWideMemoField.Create . . . 645
16.65.5 TWideMemoField.Value . . . 646
16.66 TWideStringField . . . 646
16.66.1 Description . . . 646
16.66.2 Method overview . . . 646
16.66.3 Property overview . . . 646
16.66.4 TWideStringField.Create . . . 646
16.66.5 TWideStringField.SetFieldType . . . 646
16.66.6 TWideStringField.Value . . . 647
16.67 TWordField . . . 647
16.67.1 Description . . . 647

16.67.2 Method overview . . . 647
16.67.3 TWordField.Create . . . 647

17 Reference for unit 'dbugintf' 648
17.1 Used units . . . 648
17.2 Overview . . . 648
17.3 Writing a debug server. . . . 648
17.4 Constants, types and variables . . . 649
17.4.1 Resource strings . . . 649
17.4.2 Types . . . 649
17.4.3 Variables . . . 649
17.5 Procedures and functions . . . 650
17.5.1 FreeDebugClient . . . 650
17.5.2 GetDebuggingEnabled . . . 650
17.5.3 InitDebugClient . . . 650
17.5.4 SendBoolean . . . 651
17.5.5 SendDateTime . . . 651
17.5.6 SendDebug . . . 651
17.5.7 SendDebugEx . . . 651
17.5.8 SendDebugFmt . . . 652
17.5.9 SendDebugFmtEx . . . 652
17.5.10 SendInteger . . . 652
17.5.11 SendMethodEnter . . . 653
17.5.12 SendMethodExit . . . 653
17.5.13 SendPointer . . . 653
17.5.14 SendSeparator . . . 654
17.5.15 SetDebuggingEnabled . . . 654
17.5.16 StartDebugServer . . . 654

18 Reference for unit 'dbugmsg' 655
18.1 Used units . . . 655
18.2 Overview . . . 655
18.3 Constants, types and variables . . . 655
18.3.1 Constants . . . 655
18.3.2 Types . . . 656
18.4 Procedures and functions . . . 656
18.4.1 DebugMessageName . . . 656
18.4.2 ReadDebugMessageFromStream . . . 656
18.4.3 WriteDebugMessageToStream . . . 656
18.5 TDebugMessage . . . 657

19 Reference for unit 'eventlog' 658
19.1 Used units . . . 658
19.2 Overview . . . 658
19.3 Constants, types and variables . . . 658
19.3.1 Resource strings . . . 658
19.3.2 Constants . . . 659
19.3.3 Types . . . 659
19.4 ELogError . . . 660
19.4.1 Description . . . 660
19.5 TEventLog . . . 660
19.5.1 Description . . . 660
19.5.2 Method overview . . . 660
19.5.3 Property overview . . . 661
19.5.4 TEventLog.Create . . . 661
19.5.5 TEventLog.Destroy . . . 661
19.5.6 TEventLog.EventTypeToString . . . 661
19.5.7 TEventLog.RegisterMessageFile . . . 662
19.5.8 TEventLog.UnRegisterMessageFile . . . 662
19.5.9 TEventLog.Pause . . . 663
19.5.10 TEventLog.Resume . . . 663
19.5.11 TEventLog.Log . . . 663
19.5.12 TEventLog.Warning . . . 663
19.5.13 TEventLog.Error . . . 664
19.5.14 TEventLog.Debug . . . 664
19.5.15 TEventLog.Info . . . 664
19.5.16 TEventLog.AppendContent . . . 664
19.5.17 TEventLog.Identification . . . 665
19.5.18 TEventLog.EventFilter . . . 665
19.5.19 TEventLog.LogType . . . 665
19.5.20 TEventLog.Active . . . 665
19.5.21 TEventLog.RaiseExceptionOnError . . . 666
19.5.22 TEventLog.DefaultEventType . . . 666
19.5.23 TEventLog.FileName . . . 666
19.5.24 TEventLog.TimeStampFormat . . . 667
19.5.25 TEventLog.CustomLogType . . . 667
19.5.26 TEventLog.EventIDOffset . . . 667
19.5.27 TEventLog.OnGetCustomCategory . . . 668
19.5.28 TEventLog.OnGetCustomEventID . . . 668
19.5.29 TEventLog.OnGetCustomEvent . . . 668
19.5.30 TEventLog.OnLogMessage . . . 668

19.5.31 TEventLog.Paused 669

20 Reference for unit 'ezcgi' 670
20.1 Used units 670
20.2 Overview 670
20.3 Constants, types and variables 670
20.3.1 Constants 670
20.4 ECGIException 670
20.4.1 Description 670
20.5 TEZcgi 671
20.5.1 Description 671
20.5.2 Method overview 671
20.5.3 Property overview 671
20.5.4 TEZcgi.Create 671
20.5.5 TEZcgi.Destroy 671
20.5.6 TEZcgi.Run 672
20.5.7 TEZcgi.WriteContent 672
20.5.8 TEZcgi.PutLine 672
20.5.9 TEZcgi.GetValue 673
20.5.10 TEZcgi.DoPost 673
20.5.11 TEZcgi.DoGet 673
20.5.12 TEZcgi.Values 673
20.5.13 TEZcgi.Names 674
20.5.14 TEZcgi.Variables 674
20.5.15 TEZcgi.VariableCount 675
20.5.16 TEZcgi.Name 675
20.5.17 TEZcgi.Email 675

21 Reference for unit 'FileInfo' 676
21.1 Used units 676
21.2 Overview 676
21.3 Constants, types and variables 676
21.3.1 Types 676
21.4 Procedures and functions 677
21.4.1 assign(TProgramVersion):TVersionQuad 677
21.4.2 assign(TVersionQuad):TProgramVersion 677
21.4.3 CompareProgramVersion 677
21.4.4 CompareVersionQuads 678
21.4.5 GetProgramVersion 678
21.4.6 NewerVersion 678
21.4.7 ProgramVersionToStr 678

21.4.8 StrToProgramVersion . . . 679
21.4.9 StrToVersionQuad . . . 679
21.4.10 TryStrToProgramVersion . . . 679
21.4.11 TryStrToVersionQuad . . . 679
21.4.12 VersionQuadToStr . . . 680
21.5 TProgramVersion . . . 680
21.6 EVersionInfo . . . 680
21.6.1 Description . . . 680
21.7 TFileVersionInfo . . . 680
21.7.1 Description . . . 680
21.7.2 Method overview . . . 680
21.7.3 Property overview . . . 681
21.7.4 TFileVersionInfo.Create . . . 681
21.7.5 TFileVersionInfo.Destroy . . . 681
21.7.6 TFileVersionInfo.Loaded . . . 681
21.7.7 TFileVersionInfo.ReadFileInfo . . . 681
21.7.8 TFileVersionInfo.Enabled . . . 682
21.7.9 TFileVersionInfo.FileName . . . 682
21.7.10 TFileVersionInfo.VersionStrings . . . 682
21.7.11 TFileVersionInfo.Filter . . . 683
21.7.12 TFileVersionInfo.Translation . . . 683
21.7.13 TFileVersionInfo.TranslationOnly . . . 683
21.8 TVersionInfo . . . 683
21.8.1 Description . . . 683
21.8.2 Method overview . . . 684
21.8.3 Property overview . . . 684
21.8.4 TVersionInfo.Create . . . 684
21.8.5 TVersionInfo.Destroy . . . 684
21.8.6 TVersionInfo.Load . . . 684
21.8.7 TVersionInfo.FixedInfo . . . 685
21.8.8 TVersionInfo.StringFileInfo . . . 685
21.8.9 TVersionInfo.VarFileInfo . . . 685

22 Reference for unit 'fpexprpars' 686
22.1 Used units . . . 686
22.2 Overview . . . 686
22.3 Constants, types and variables . . . 687
22.3.1 Constants . . . 687
22.3.2 Types . . . 687
22.4 Procedures and functions . . . 693

22.4.1 ArgToFloat . . . 693
22.4.2 BuiltinIdentifiers . . . 693
22.4.3 CharToResultType . . . 693
22.4.4 RegisterStdBuiltins . . . 694
22.4.5 ResultTypeName . . . 694
22.4.6 TokenName . . . 694
22.5 TFPExpressionResult . . . 694
22.6 EExprParser . . . 695
22.6.1 Description . . . 695
22.7 EExprScanner . . . 695
22.7.1 Description . . . 695
22.8 TAggregateAvg . . . 696
22.8.1 Description . . . 696
22.8.2 Method overview . . . 696
22.8.3 TAggregateAvg.InitAggregate . . . 696
22.8.4 TAggregateAvg.UpdateAggregate . . . 697
22.8.5 TAggregateAvg.GetNodeValue . . . 697
22.9 TAggregateCount . . . 697
22.9.1 Description . . . 697
22.9.2 Method overview . . . 697
22.9.3 TAggregateCount.InitAggregate . . . 698
22.9.4 TAggregateCount.UpdateAggregate . . . 698
22.10TAggregateExpr . . . 698
22.10.1 Description . . . 698
22.10.2 Method overview . . . 698
22.10.3 TAggregateExpr.IsAggregate . . . 699
22.10.4 TAggregateExpr.GetNodeValue . . . 699
22.11TAggregateMax . . . 699
22.11.1 Description . . . 699
22.11.2 Method overview . . . 699
22.11.3 TAggregateMax.InitAggregate . . . 700
22.11.4 TAggregateMax.UpdateAggregate . . . 700
22.12TAggregateMin . . . 700
22.12.1 Description . . . 700
22.12.2 Method overview . . . 700
22.12.3 TAggregateMin.InitAggregate . . . 701
22.12.4 TAggregateMin.UpdateAggregate . . . 701
22.13TAggregateSum . . . 701
22.13.1 Description . . . 701
22.13.2 Method overview . . . 701

22.13.3 TAggregateSum.ConvertArgument . . . 702
22.13.4 TAggregateSum.InitAggregate . . . 702
22.13.5 TAggregateSum.UpdateAggregate . . . 702
22.14 TCaseOperation . . . 702
22.14.1 Description . . . 702
22.14.2 Method overview . . . 703
22.14.3 Property overview . . . 703
22.14.4 TCaseOperation.Check . . . 703
22.14.5 TCaseOperation.InitAggregate . . . 703
22.14.6 TCaseOperation.UpdateAggregate . . . 704
22.14.7 TCaseOperation.HasAggregate . . . 704
22.14.8 TCaseOperation.NodeType . . . 704
22.14.9 TCaseOperation.Create . . . 704
22.14.10 TCaseOperation.destroy . . . 705
22.14.11 TCaseOperation.AsString . . . 705
22.14.12 TCaseOperation.Condition . . . 705
22.15 TCurrencyToDateTimeNode . . . 706
22.15.1 Description . . . 706
22.15.2 Method overview . . . 706
22.15.3 TCurrencyToDateTimeNode.Check . . . 706
22.15.4 TCurrencyToDateTimeNode.NodeType . . . 706
22.15.5 TCurrencyToDateTimeNode.GetNodeValue . . . 707
22.16 TCurrencyToFloatNode . . . 707
22.16.1 Description . . . 707
22.16.2 Method overview . . . 707
22.16.3 TCurrencyToFloatNode.Check . . . 707
22.16.4 TCurrencyToFloatNode.NodeType . . . 707
22.16.5 TCurrencyToFloatNode.GetNodeValue . . . 708
22.17 TExprBuiltInManager . . . 708
22.17.1 Description . . . 708
22.17.2 Method overview . . . 708
22.17.3 Property overview . . . 708
22.17.4 TExprBuiltInManager.Create . . . 709
22.17.5 TExprBuiltInManager.Destroy . . . 709
22.17.6 TExprBuiltInManager.IndexOfIdentifier . . . 709
22.17.7 TExprBuiltInManager.FindIdentifier . . . 709
22.17.8 TExprBuiltInManager.IdentifierByName . . . 709
22.17.9 TExprBuiltInManager.AddVariable . . . 710
22.17.10 TExprBuiltInManager.AddBooleanVariable . . . 710
22.17.11 TExprBuiltInManager.AddIntegerVariable . . . 710

22.17.12TExprBuiltInManager.AddFloatVariable 710
22.17.13TExprBuiltInManager.AddCurrencyVariable 711
22.17.14TExprBuiltInManager.AddStringVariable 711
22.17.15TExprBuiltInManager.AddDateTimeVariable 711
22.17.16TExprBuiltInManager.AddFunction 711
22.17.17TExprBuiltInManager.Delete 712
22.17.18TExprBuiltInManager.Remove 712
22.17.19TExprBuiltInManager.IdentifierCount 712
22.17.20TExprBuiltInManager.Identifiers 712
22.18TFloatToCurrencyNode 713
22.18.1 Description 713
22.18.2 Method overview 713
22.18.3 TFloatToCurrencyNode.Check 713
22.18.4 TFloatToCurrencyNode.NodeType 713
22.18.5 TFloatToCurrencyNode.GetNodeValue 714
22.19TFloatToDateTimeNode 714
22.19.1 Description 714
22.19.2 Method overview 714
22.19.3 TFloatToDateTimeNode.Check 714
22.19.4 TFloatToDateTimeNode.NodeType 714
22.19.5 TFloatToDateTimeNode.GetNodeValue 715
22.20TFPAddOperation 715
22.20.1 Description 715
22.20.2 Method overview 715
22.20.3 TFPAddOperation.AsString 715
22.21TFPBinaryAndOperation 716
22.21.1 Description 716
22.21.2 Method overview 716
22.21.3 TFPBinaryAndOperation.AsString 716
22.22TFPBinaryOperation 716
22.22.1 Description 716
22.22.2 Method overview 717
22.22.3 Property overview 717
22.22.4 TFPBinaryOperation.Create 717
22.22.5 TFPBinaryOperation.Destroy 717
22.22.6 TFPBinaryOperation.InitAggregate 717
22.22.7 TFPBinaryOperation.UpdateAggregate 718
22.22.8 TFPBinaryOperation.HasAggregate 718
22.22.9 TFPBinaryOperation.Check 718
22.22.10TFPBinaryOperation.left 718

22.22.1 TFPBinaryOperation.Right . . . 719
22.23 TFPBinaryOrOperation . . . 719
22.23.1 Description . . . 719
22.23.2 Method overview . . . 719
22.23.3 TFPBinaryOrOperation.AsString . . . 719
22.24 TFPBinaryXOrOperation . . . 720
22.24.1 Description . . . 720
22.24.2 Method overview . . . 720
22.24.3 TFPBinaryXOrOperation.AsString . . . 720
22.25 TFPBooleanOperation . . . 720
22.25.1 Description . . . 720
22.25.2 Method overview . . . 721
22.25.3 TFPBooleanOperation.Check . . . 721
22.25.4 TFPBooleanOperation.NodeType . . . 721
22.26 TFPBooleanResultOperation . . . 721
22.26.1 Description . . . 721
22.26.2 Method overview . . . 722
22.26.3 TFPBooleanResultOperation.Check . . . 722
22.26.4 TFPBooleanResultOperation.NodeType . . . 722
22.27 TFPBuiltInExprIdentifierDef . . . 722
22.27.1 Description . . . 722
22.27.2 Method overview . . . 722
22.27.3 Property overview . . . 722
22.27.4 TFPBuiltInExprIdentifierDef.Assign . . . 723
22.27.5 TFPBuiltInExprIdentifierDef.Category . . . 723
22.28 TFPConstExpression . . . 723
22.28.1 Description . . . 723
22.28.2 Method overview . . . 723
22.28.3 Property overview . . . 724
22.28.4 TFPConstExpression.CreateString . . . 724
22.28.5 TFPConstExpression.CreateInteger . . . 724
22.28.6 TFPConstExpression.CreateDateTime . . . 724
22.28.7 TFPConstExpression.CreateFloat . . . 724
22.28.8 TFPConstExpression.CreateBoolean . . . 725
22.28.9 TFPConstExpression.CreateCurrency . . . 725
22.28.10 TFPConstExpression.Check . . . 725
22.28.11 TFPConstExpression.NodeType . . . 725
22.28.12 TFPConstExpression.GetNodeValue . . . 725
22.28.13 TFPConstExpression.AsString . . . 726
22.28.14 TFPConstExpression.ConstValue . . . 726

22.29TFPConvertNode . . . 726
22.29.1 Description . . . 726
22.29.2 Method overview . . . 726
22.29.3 TFPConvertNode.AsString . . . 726
22.30TFPDivideOperation . . . 727
22.30.1 Description . . . 727
22.30.2 Method overview . . . 727
22.30.3 TFPDivideOperation.Check . . . 727
22.30.4 TFPDivideOperation.AsString . . . 727
22.30.5 TFPDivideOperation.NodeType . . . 727
22.30.6 TFPDivideOperation.GetNodeValue . . . 728
22.31TFPEqualOperation . . . 728
22.31.1 Description . . . 728
22.31.2 Method overview . . . 728
22.31.3 TFPEqualOperation.AsString . . . 728
22.32TFPExpressionParser . . . 728
22.32.1 Description . . . 728
22.32.2 Method overview . . . 729
22.32.3 Property overview . . . 729
22.32.4 TFPExpressionParser.Create . . . 729
22.32.5 TFPExpressionParser.Destroy . . . 729
22.32.6 TFPExpressionParser.IdentifierByName . . . 730
22.32.7 TFPExpressionParser.Clear . . . 730
22.32.8 TFPExpressionParser.EvaluateExpression . . . 730
22.32.9 TFPExpressionParser.ExtractIdentifierNames . . . 730
22.32.10TFPExpressionParser.ExtractNode . . . 730
22.32.11TFPExpressionParser.Evaluate . . . 731
22.32.12TFPExpressionParser.ResultType . . . 731
22.32.13TFPExpressionParser.HasAggregate . . . 731
22.32.14TFPExpressionParser.InitAggregate . . . 731
22.32.15TFPExpressionParser.UpdateAggregate . . . 732
22.32.16TFPExpressionParser.AsFloat . . . 732
22.32.17TFPExpressionParser.AsCurrency . . . 732
22.32.18TFPExpressionParser.AsInteger . . . 732
22.32.19TFPExpressionParser.AsString . . . 733
22.32.20TFPExpressionParser.AsBoolean . . . 733
22.32.21TFPExpressionParser.AsDateTime . . . 733
22.32.22TFPExpressionParser.Expression . . . 733
22.32.23TFPExpressionParser.Identifiers . . . 734
22.32.24TFPExpressionParser.BuiltIns . . . 734

22.33TFPExpressionScanner . . . 734
22.33.1 Description . . . 734
22.33.2 Method overview . . . 734
22.33.3 Property overview . . . 735
22.33.4 TFPExpressionScanner.Create . . . 735
22.33.5 TFPExpressionScanner.GetToken . . . 735
22.33.6 TFPExpressionScanner.Token . . . 735
22.33.7 TFPExpressionScanner.TokenType . . . 735
22.33.8 TFPExpressionScanner.Source . . . 736
22.33.9 TFPExpressionScanner.Pos . . . 736
22.33.10TFPExpressionScanner.CurrentChar . . . 736
22.34TFPExprFunction . . . 737
22.34.1 Description . . . 737
22.34.2 Method overview . . . 737
22.34.3 Property overview . . . 737
22.34.4 TFPExprFunction.Check . . . 737
22.34.5 TFPExprFunction.CreateFunction . . . 737
22.34.6 TFPExprFunction.Destroy . . . 738
22.34.7 TFPExprFunction.InitAggregate . . . 738
22.34.8 TFPExprFunction.UpdateAggregate . . . 738
22.34.9 TFPExprFunction.HasAggregate . . . 738
22.34.10TFPExprFunction.AsString . . . 739
22.34.11TFPExprFunction.ArgumentNodes . . . 739
22.34.12TFPExprFunction.ArgumentParams . . . 739
22.35TFPExprIdentifierDef . . . 739
22.35.1 Description . . . 739
22.35.2 Method overview . . . 739
22.35.3 Property overview . . . 740
22.35.4 TFPExprIdentifierDef.ArgumentCount . . . 740
22.35.5 TFPExprIdentifierDef.Assign . . . 740
22.35.6 TFPExprIdentifierDef.EventBasedVariable . . . 740
22.35.7 TFPExprIdentifierDef.AsFloat . . . 741
22.35.8 TFPExprIdentifierDef.AsCurrency . . . 741
22.35.9 TFPExprIdentifierDef.AsInteger . . . 741
22.35.10TFPExprIdentifierDef.AsString . . . 741
22.35.11TFPExprIdentifierDef.AsBoolean . . . 742
22.35.12TFPExprIdentifierDef.AsDateTime . . . 742
22.35.13TFPExprIdentifierDef.OnGetFunctionValueCallBack . . . 742
22.35.14TFPExprIdentifierDef.OnGetVariableValueCallBack . . . 742
22.35.15TFPExprIdentifierDef.IdentifierType . . . 743

22.35.16TFPExprIdentifierDef.Name . . . 743
22.35.17TFPExprIdentifierDef.Value . . . 743
22.35.18TFPExprIdentifierDef.ParameterTypes . . . 743
22.35.19TFPExprIdentifierDef.ResultType . . . 744
22.35.20TFPExprIdentifierDef.OnGetFunctionValue . . . 744
22.35.21TFPExprIdentifierDef.OnGetVariableValue . . . 744
22.35.22TFPExprIdentifierDef.NodeType . . . 744
22.35.23TFPExprIdentifierDef.VariableArgumentCount . . . 745
22.36TFPExprIdentifierDefs . . . 745
22.36.1 Description . . . 745
22.36.2 Method overview . . . 745
22.36.3 Property overview . . . 745
22.36.4 TFPExprIdentifierDefs.IndexOfIdentifier . . . 745
22.36.5 TFPExprIdentifierDefs.FindIdentifier . . . 746
22.36.6 TFPExprIdentifierDefs.IdentifierByName . . . 746
22.36.7 TFPExprIdentifierDefs.AddVariable . . . 746
22.36.8 TFPExprIdentifierDefs.AddBooleanVariable . . . 746
22.36.9 TFPExprIdentifierDefs.AddIntegerVariable . . . 747
22.36.10TFPExprIdentifierDefs.AddFloatVariable . . . 747
22.36.11TFPExprIdentifierDefs.AddCurrencyVariable . . . 747
22.36.12TFPExprIdentifierDefs.AddStringVariable . . . 747
22.36.13TFPExprIdentifierDefs.AddDateTimeVariable . . . 748
22.36.14TFPExprIdentifierDefs.AddFunction . . . 748
22.36.15TFPExprIdentifierDefs.Identifiers . . . 748
22.37TFPExprIdentifierNode . . . 749
22.37.1 Description . . . 749
22.37.2 Method overview . . . 749
22.37.3 Property overview . . . 749
22.37.4 TFPExprIdentifierNode.CreateIdentifier . . . 749
22.37.5 TFPExprIdentifierNode.NodeType . . . 749
22.37.6 TFPExprIdentifierNode.GetNodeValue . . . 750
22.37.7 TFPExprIdentifierNode.Identifier . . . 750
22.38TFPExprNode . . . 750
22.38.1 Description . . . 750
22.38.2 Method overview . . . 750
22.38.3 TFPExprNode.Check . . . 751
22.38.4 TFPExprNode.InitAggregate . . . 751
22.38.5 TFPExprNode.UpdateAggregate . . . 751
22.38.6 TFPExprNode.IsAggregate . . . 751
22.38.7 TFPExprNode.HasAggregate . . . 752

22.38.8 TFPExprNode.NodeType . . . 752
22.38.9 TFPExprNode.NodeValue . . . 752
22.38.10TFPExprNode.AsString . . . 753
22.39TFPExprVariable . . . 753
22.39.1 Description . . . 753
22.39.2 Method overview . . . 753
22.39.3 TFPExprVariable.Check . . . 753
22.39.4 TFPExprVariable.AsString . . . 754
22.40TFPFunctionCallBack . . . 754
22.40.1 Description . . . 754
22.40.2 Method overview . . . 754
22.40.3 Property overview . . . 754
22.40.4 TFPFunctionCallBack.CreateFunction . . . 754
22.40.5 TFPFunctionCallBack.GetNodeValue . . . 755
22.40.6 TFPFunctionCallBack.CallBack . . . 755
22.41TFPFunctionEventHandler . . . 755
22.41.1 Description . . . 755
22.41.2 Method overview . . . 755
22.41.3 Property overview . . . 755
22.41.4 TFPFunctionEventHandler.CreateFunction . . . 755
22.41.5 TFPFunctionEventHandler.GetNodeValue . . . 756
22.41.6 TFPFunctionEventHandler.CallBack . . . 756
22.42TFPGreaterThanEqualOperation . . . 756
22.42.1 Description . . . 756
22.42.2 Method overview . . . 756
22.42.3 TFPGreaterThanEqualOperation.AsString . . . 757
22.43TFPGreaterThanOperation . . . 757
22.43.1 Description . . . 757
22.43.2 Method overview . . . 757
22.43.3 TFPGreaterThanOperation.AsString . . . 757
22.44TFPLessThanEqualOperation . . . 758
22.44.1 Description . . . 758
22.44.2 Method overview . . . 758
22.44.3 TFPLessThanEqualOperation.AsString . . . 758
22.45TFPLessThanOperation . . . 758
22.45.1 Description . . . 758
22.45.2 Method overview . . . 759
22.45.3 TFPLessThanOperation.AsString . . . 759
22.46TFPModuloOperation . . . 759
22.46.1 Description . . . 759

22.46.2 Method overview . . . 759
22.46.3 TFPModuloOperation.Check . . . 759
22.46.4 TFPModuloOperation.AsString . . . 759
22.46.5 TFPModuloOperation.NodeType . . . 760
22.46.6 TFPModuloOperation.GetNodeValue . . . 760
22.47TFPMultiplyOperation . . . 760
22.47.1 Description . . . 760
22.47.2 Method overview . . . 760
22.47.3 TFPMultiplyOperation.check . . . 760
22.47.4 TFPMultiplyOperation.AsString . . . 761
22.47.5 TFPMultiplyOperation.GetNodeValue . . . 761
22.48TFPNegateOperation . . . 761
22.48.1 Description . . . 761
22.48.2 Method overview . . . 761
22.48.3 TFPNegateOperation.Check . . . 761
22.48.4 TFPNegateOperation.NodeType . . . 762
22.48.5 TFPNegateOperation.GetNodeValue . . . 762
22.48.6 TFPNegateOperation.AsString . . . 762
22.49TFPNotNode . . . 762
22.49.1 Description . . . 762
22.49.2 Method overview . . . 763
22.49.3 TFPNotNode.Check . . . 763
22.49.4 TFPNotNode.NodeType . . . 763
22.49.5 TFPNotNode.GetNodeValue . . . 763
22.49.6 TFPNotNode.AsString . . . 763
22.50TFPOrderingOperation . . . 764
22.50.1 Description . . . 764
22.50.2 Method overview . . . 764
22.50.3 TFPOrderingOperation.Check . . . 764
22.51TFPPowerOperation . . . 764
22.51.1 Description . . . 764
22.51.2 Method overview . . . 765
22.51.3 TFPPowerOperation.Check . . . 765
22.51.4 TFPPowerOperation.AsString . . . 765
22.51.5 TFPPowerOperation.NodeType . . . 765
22.51.6 TFPPowerOperation.GetNodeValue . . . 766
22.52TFPSubtractOperation . . . 766
22.52.1 Description . . . 766
22.52.2 Method overview . . . 766
22.52.3 TFPSubtractOperation.Check . . . 766

22.52.4 TFPSubtractOperation.GetNodeValue . . . 766
22.52.5 TFPSubtractOperation.AsString . . . 767
22.53TFPUnaryOperator . . . 767
22.53.1 Description . . . 767
22.53.2 Method overview . . . 767
22.53.3 Property overview . . . 767
22.53.4 TFPUnaryOperator.Create . . . 767
22.53.5 TFPUnaryOperator.Destroy . . . 768
22.53.6 TFPUnaryOperator.InitAggregate . . . 768
22.53.7 TFPUnaryOperator.UpdateAggregate . . . 768
22.53.8 TFPUnaryOperator.HasAggregate . . . 769
22.53.9 TFPUnaryOperator.Check . . . 769
22.53.10TFPUnaryOperator.Operand . . . 769
22.54TFPUnequalOperation . . . 769
22.54.1 Description . . . 769
22.54.2 Method overview . . . 770
22.54.3 TFPUnequalOperation.AsString . . . 770
22.55TIfOperation . . . 770
22.55.1 Description . . . 770
22.55.2 Method overview . . . 770
22.55.3 Property overview . . . 770
22.55.4 TIfOperation.Check . . . 771
22.55.5 TIfOperation.InitAggregate . . . 771
22.55.6 TIfOperation.UpdateAggregate . . . 771
22.55.7 TIfOperation.HasAggregate . . . 771
22.55.8 TIfOperation.NodeType . . . 772
22.55.9 TIfOperation.Create . . . 772
22.55.10TIfOperation.destroy . . . 772
22.55.11TIfOperation.AsString . . . 772
22.55.12TIfOperation.Condition . . . 773
22.56TIntConvertNode . . . 773
22.56.1 Description . . . 773
22.56.2 Method overview . . . 773
22.56.3 TIntConvertNode.Check . . . 773
22.57TIntToCurrencyNode . . . 774
22.57.1 Description . . . 774
22.57.2 Method overview . . . 774
22.57.3 TIntToCurrencyNode.NodeType . . . 774
22.57.4 TIntToCurrencyNode.GetNodeValue . . . 774
22.58TIntToDateTimeNode . . . 775

22.58.1 Description . . . 775
22.58.2 Method overview . . . 775
22.58.3 TIntToDateTimeNode.NodeType . . . 775
22.58.4 TIntToDateTimeNode.GetNodeValue . . . 775
22.59TIntToFloatNode . . . 775
22.59.1 Description . . . 775
22.59.2 Method overview . . . 776
22.59.3 TIntToFloatNode.NodeType . . . 776
22.59.4 TIntToFloatNode.GetNodeValue . . . 776
22.60TMathOperation . . . 776
22.60.1 Description . . . 776
22.60.2 Method overview . . . 777
22.60.3 TMathOperation.Check . . . 777
22.60.4 TMathOperation.NodeType . . . 777

23 Reference for unit 'fphttp' 778
23.1 Used units . . . 778
23.2 Overview . . . 778
23.3 Constants, types and variables . . . 778
23.3.1 Resource strings . . . 778
23.3.2 Types . . . 779
23.3.3 Variables . . . 780
23.4 Procedures and functions . . . 780
23.4.1 RegisterHTTPModule . . . 780
23.4.2 SessionFactory . . . 781
23.5 EFPHTTPError . . . 781
23.5.1 Description . . . 781
23.6 EWebSessionError . . . 781
23.6.1 Description . . . 781
23.7 TCustomHTTPModule . . . 781
23.7.1 Description . . . 781
23.7.2 Method overview . . . 781
23.7.3 Property overview . . . 782
23.7.4 TCustomHTTPModule.CreateNew . . . 782
23.7.5 TCustomHTTPModule.RegisterModule . . . 782
23.7.6 TCustomHTTPModule.HandleRequest . . . 782
23.7.7 TCustomHTTPModule.DoAfterInitModule . . . 782
23.7.8 TCustomHTTPModule.Kind . . . 783
23.7.9 TCustomHTTPModule.BaseURL . . . 783
23.7.10 TCustomHTTPModule.AfterInitModule . . . 783

23.8 TCustomWebAction . . . 783
23.8.1 Description . . . 783
23.8.2 Method overview . . . 783
23.8.3 Property overview . . . 784
23.8.4 TCustomWebAction.GetNamePath . . . 784
23.8.5 TCustomWebAction.Name . . . 784
23.8.6 TCustomWebAction.ContentProducer . . . 784
23.8.7 TCustomWebAction.Default . . . 784
23.8.8 TCustomWebAction.BeforeRequest . . . 785
23.8.9 TCustomWebAction.AfterResponse . . . 785
23.9 TCustomWebActions . . . 785
23.9.1 Description . . . 785
23.9.2 Method overview . . . 785
23.9.3 Property overview . . . 785
23.9.4 TCustomWebActions.Create . . . 785
23.9.5 TCustomWebActions.Assign . . . 786
23.9.6 TCustomWebActions.Add . . . 786
23.9.7 TCustomWebActions.ActionByName . . . 786
23.9.8 TCustomWebActions.FindAction . . . 786
23.9.9 TCustomWebActions.IndexOfAction . . . 786
23.9.10 TCustomWebActions.OnGetAction . . . 786
23.9.11 TCustomWebActions.Actions . . . 787
23.9.12 TCustomWebActions.DefActionWhenUnknown . . . 787
23.10THTTPContentProducer . . . 787
23.10.1 Description . . . 787
23.10.2 Method overview . . . 787
23.10.3 Property overview . . . 787
23.10.4 THTTPContentProducer.HandleRequest . . . 787
23.10.5 THTTPContentProducer.GetContent . . . 788
23.10.6 THTTPContentProducer.HaveContent . . . 788
23.10.7 THTTPContentProducer.ContentToStream . . . 788
23.10.8 THTTPContentProducer.Request . . . 788
23.10.9 THTTPContentProducer.Response . . . 788
23.11TModuleFactory . . . 789
23.11.1 Description . . . 789
23.11.2 Method overview . . . 789
23.11.3 Property overview . . . 789
23.11.4 TModuleFactory.RegisterHTTPModule . . . 789
23.11.5 TModuleFactory.FindModule . . . 789
23.11.6 TModuleFactory.ModuleByName . . . 789

23.11.7 TModuleFactory.IndexOfModule . . . 790
23.11.8 TModuleFactory.Modules . . . 790
23.11.9 TModuleFactory.OnModuleRequest . . . 790
23.12TModuleItem . . . 790
23.12.1 Description . . . 790
23.12.2 Interfaces overview . . . 790
23.12.3 Method overview . . . 790
23.12.4 Property overview . . . 790
23.12.5 TModuleItem.Destroy . . . 791
23.12.6 TModuleItem.ModuleClass . . . 791
23.12.7 TModuleItem.ModuleName . . . 791
23.12.8 TModuleItem.SkipStreaming . . . 791
23.13TSessionFactory . . . 791
23.13.1 Description . . . 791
23.13.2 Method overview . . . 792
23.13.3 Property overview . . . 792
23.13.4 TSessionFactory.CreateSession . . . 792
23.13.5 TSessionFactory.DoneSession . . . 792
23.13.6 TSessionFactory.CleanupSessions . . . 792
23.13.7 TSessionFactory.CleanupInterval . . . 792
23.13.8 TSessionFactory.DefaultTimeOutMinutes . . . 793
23.13.9 TSessionFactory.SessionCookie . . . 793
23.13.10TSessionFactory.SessionCookiePath . . . 793
23.14TSessionHTTPModule . . . 793
23.14.1 Description . . . 793
23.14.2 Method overview . . . 793
23.14.3 Property overview . . . 793
23.14.4 TSessionHTTPModule.destroy . . . 794
23.14.5 TSessionHTTPModule.HandleRequest . . . 794
23.14.6 TSessionHTTPModule.CreateSession . . . 794
23.14.7 TSessionHTTPModule.Session . . . 794
23.14.8 TSessionHTTPModule.OnNewSession . . . 794
23.14.9 TSessionHTTPModule.OnSessionExpired . . . 795

24 Reference for unit 'fphttpclient' 796

24.1 Used units . . . 796
24.2 Overview . . . 796
24.3 Advanced TFPHTTPClient Features . . . 796
24.4 File Upload with Local Filename . . . 798
24.5 Form Data and File Upload with fpHttpClient . . . 800

24.6 HTTP Methods with TFPHTTPClient . . . 801
24.7 HTTPS Support in TFPHTTPClient . . . 803
24.8 Using the TFPHTTPClient . . . 804
24.9 Constants, types and variables . . . 805
24.9.1 Constants . . . 805
24.9.2 Types . . . 805
24.10 Procedures and functions . . . 806
24.10.1 DecodeURLElement . . . 806
24.10.2 EncodeURLElement . . . 806
24.11 EHTTPClient . . . 806
24.11.1 Description . . . 806
24.12 TFPCustomHTTPClient . . . 806
24.12.1 Description . . . 806
24.12.2 Method overview . . . 807
24.12.3 Property overview . . . 807
24.12.4 TFPCustomHTTPClient.Create . . . 807
24.12.5 TFPCustomHTTPClient.Destroy . . . 807
24.12.6 TFPCustomHTTPClient.AddHeader . . . 808
24.12.7 TFPCustomHTTPClient.IndexOfHeader . . . 808
24.12.8 TFPCustomHTTPClient.GetHeader . . . 808
24.12.9 TFPCustomHTTPClient.Terminate . . . 809
24.12.10 TFPCustomHTTPClient.HTTPMethod . . . 809
24.12.11 TFPCustomHTTPClient.Get . . . 809
24.12.12 TFPCustomHTTPClient.IsRedirect . . . 809
24.12.13 TFPCustomHTTPClient.RedirectForcesGET . . . 810
24.12.14 TFPCustomHTTPClient.SimpleGet . . . 810
24.12.15 TFPCustomHTTPClient.Post . . . 810
24.12.16 TFPCustomHTTPClient.SimplePost . . . 810
24.12.17 TFPCustomHTTPClient.Put . . . 811
24.12.18 TFPCustomHTTPClient.SimplePut . . . 811
24.12.19 TFPCustomHTTPClient.Delete . . . 811
24.12.20 TFPCustomHTTPClient.SimpleDelete . . . 812
24.12.21 TFPCustomHTTPClient.Patch . . . 812
24.12.22 TFPCustomHTTPClient.SimplePatch . . . 812
24.12.23 TFPCustomHTTPClient.Options . . . 813
24.12.24 TFPCustomHTTPClient.SimpleOptions . . . 813
24.12.25 TFPCustomHTTPClient.Head . . . 813
24.12.26 TFPCustomHTTPClient.FormPost . . . 814
24.12.27 TFPCustomHTTPClient.SimpleFormPost . . . 814
24.12.28 TFPCustomHTTPClient.FileFormPost . . . 815

24.12.29TFPCustomHTTPClient.StreamFormPost . . . 815
24.12.30TFPCustomHTTPClient.SimpleFileFormPost . . . 815
24.12.31TFPCustomHTTPClient.Terminated . . . 816
24.13TFPHTTPClient . . . 816
24.13.1 Description . . . 816
24.13.2 Property overview . . . 817
24.13.3 TFPHTTPClient.KeepConnection . . . 817
24.13.4 TFPHTTPClient.Connected . . . 817
24.13.5 TFPHTTPClient.IOTimeout . . . 818
24.13.6 TFPHTTPClient.ConnectTimeout . . . 818
24.13.7 TFPHTTPClient.RequestHeaders . . . 818
24.13.8 TFPHTTPClient.RequestBody . . . 819
24.13.9 TFPHTTPClient.ResponseHeaders . . . 819
24.13.10TFPHTTPClient.HTTPversion . . . 819
24.13.11TFPHTTPClient.ServerHTTPVersion . . . 819
24.13.12TFPHTTPClient.ResponseStatusCode . . . 820
24.13.13TFPHTTPClient.ResponseStatusText . . . 820
24.13.14TFPHTTPClient.Cookies . . . 820
24.13.15TFPHTTPClient.AllowRedirect . . . 820
24.13.16TFPHTTPClient.MaxRedirects . . . 821
24.13.17TFPHTTPClient.OnRedirect . . . 821
24.13.18TFPHTTPClient.UserName . . . 821
24.13.19TFPHTTPClient.Password . . . 821
24.13.20TFPHTTPClient.OnPassword . . . 822
24.13.21TFPHTTPClient.OnDataReceived . . . 822
24.13.22TFPHTTPClient.OnHeaders . . . 822
24.13.23TFPHTTPClient.OnGetSocketHandler . . . 822
24.13.24TFPHTTPClient.Proxy . . . 823
24.13.25TFPHTTPClient.VerifySSLCertificate . . . 823
24.13.26TFPHTTPClient.AfterSocketHandlerCreate . . . 823
24.13.27TFPHTTPClient.OnVerifySSLCertificate . . . 823
24.14TProxyData . . . 824
24.14.1 Description . . . 824
24.14.2 Method overview . . . 824
24.14.3 Property overview . . . 824
24.14.4 TProxyData.Assign . . . 824
24.14.5 TProxyData.Host . . . 824
24.14.6 TProxyData.Port . . . 824
24.14.7 TProxyData.UserName . . . 825
24.14.8 TProxyData.Password . . . 825

25 Reference for unit 'fpjson' 826
25.1 Used units . . . 826
25.2 Overview . . . 826
25.3 Constants, types and variables . . . 828
25.3.1 Constants . . . 828
25.3.2 Types . . . 829
25.4 Procedures and functions . . . 833
25.4.1 CreateJSON . . . 833
25.4.2 CreateJSONArray . . . 833
25.4.3 CreateJSONObject . . . 834
25.4.4 GetJSON . . . 834
25.4.5 GetJSONInstanceType . . . 834
25.4.6 GetJSONParserHandler . . . 835
25.4.7 GetJSONStringParserHandler . . . 835
25.4.8 JSONStringToString . . . 835
25.4.9 JSONTypeName . . . 835
25.4.10 SetJSONInstanceType . . . 836
25.4.11 SetJSONParserHandler . . . 836
25.4.12 SetJSONStringParserHandler . . . 836
25.4.13 StringToJSONString . . . 837
25.5 EJSON . . . 837
25.5.1 Description . . . 837
25.6 TBaseJSONEnumerator . . . 837
25.6.1 Description . . . 837
25.6.2 Method overview . . . 837
25.6.3 Property overview . . . 838
25.6.4 TBaseJSONEnumerator.GetCurrent . . . 838
25.6.5 TBaseJSONEnumerator.MoveNext . . . 838
25.6.6 TBaseJSONEnumerator.Current . . . 838
25.7 TJSONArray . . . 838
25.7.1 Description . . . 838
25.7.2 Method overview . . . 839
25.7.3 Property overview . . . 839
25.7.4 TJSONArray.Create . . . 839
25.7.5 TJSONArray.Destroy . . . 840
25.7.6 TJSONArray.JSONType . . . 840
25.7.7 TJSONArray.Clone . . . 840
25.7.8 TJSONArray.Iterate . . . 840
25.7.9 TJSONArray.IndexOf . . . 841
25.7.10 TJSONArray.GetEnumerator . . . 841

25.7.11 TJSONArray.Clear . . . 841
25.7.12 TJSONArray.Add . . . 841
25.7.13 TJSONArray.Delete . . . 842
25.7.14 TJSONArray.Exchange . . . 842
25.7.15 TJSONArray.Extract . . . 842
25.7.16 TJSONArray.Insert . . . 843
25.7.17 TJSONArray.Move . . . 843
25.7.18 TJSONArray.Remove . . . 843
25.7.19 TJSONArray.Sort . . . 844
25.7.20 TJSONArray.Items . . . 844
25.7.21 TJSONArray.Types . . . 844
25.7.22 TJSONArray.Nulls . . . 844
25.7.23 TJSONArray.Integers . . . 845
25.7.24 TJSONArray.Int64s . . . 845
25.7.25 TJSONArray.LargeInts . . . 846
25.7.26 TJSONArray.QWords . . . 846
25.7.27 TJSONArray.UnicodeStrings . . . 846
25.7.28 TJSONArray.Strings . . . 847
25.7.29 TJSONArray.Floats . . . 847
25.7.30 TJSONArray.Booleans . . . 848
25.7.31 TJSONArray.Arrays . . . 848
25.7.32 TJSONArray.Objects . . . 848
25.8 TJSONBoolean . . . 849
25.8.1 Description . . . 849
25.8.2 Method overview . . . 849
25.8.3 TJSONBoolean.Create . . . 849
25.8.4 TJSONBoolean.JSONType . . . 849
25.8.5 TJSONBoolean.Clear . . . 850
25.8.6 TJSONBoolean.Clone . . . 850
25.9 TJSONData . . . 850
25.9.1 Description . . . 850
25.9.2 Method overview . . . 851
25.9.3 Property overview . . . 851
25.9.4 TJSONData.JSONType . . . 851
25.9.5 TJSONData.Create . . . 851
25.9.6 TJSONData.Clear . . . 852
25.9.7 TJSONData.DumpJSON . . . 852
25.9.8 TJSONData.GetEnumerator . . . 852
25.9.9 TJSONData.FindPath . . . 852
25.9.10 TJSONData.GetPath . . . 854

25.9.11 TJSONData.Clone . . . 855
25.9.12 TJSONData.FormatJSON . . . 855
25.9.13 TJSONData.CompressedJSON . . . 855
25.9.14 TJSONData.Count . . . 856
25.9.15 TJSONData.Items . . . 856
25.9.16 TJSONData.Value . . . 856
25.9.17 TJSONData.AsString . . . 856
25.9.18 TJSONData.AsUnicodeString . . . 857
25.9.19 TJSONData.AsInt64 . . . 857
25.9.20 TJSONData.AsQWord . . . 857
25.9.21 TJSONData.AsLargeInt . . . 858
25.9.22 TJSONData.AsFloat . . . 858
25.9.23 TJSONData.AsInteger . . . 858
25.9.24 TJSONData.AsBoolean . . . 859
25.9.25 TJSONData.IsNull . . . 859
25.9.26 TJSONData.AsJSON . . . 859
25.10TJSONFloatNumber . . . 860
25.10.1 Description . . . 860
25.10.2 Method overview . . . 860
25.10.3 TJSONFloatNumber.Create . . . 860
25.10.4 TJSONFloatNumber.NumberType . . . 860
25.10.5 TJSONFloatNumber.Clear . . . 861
25.10.6 TJSONFloatNumber.Clone . . . 861
25.11TJSONInt64Number . . . 861
25.11.1 Description . . . 861
25.11.2 Method overview . . . 861
25.11.3 TJSONInt64Number.Create . . . 861
25.11.4 TJSONInt64Number.NumberType . . . 862
25.11.5 TJSONInt64Number.Clear . . . 862
25.11.6 TJSONInt64Number.Clone . . . 862
25.12TJSONIntegerNumber . . . 862
25.12.1 Description . . . 862
25.12.2 Method overview . . . 862
25.12.3 TJSONIntegerNumber.Create . . . 863
25.12.4 TJSONIntegerNumber.NumberType . . . 863
25.12.5 TJSONIntegerNumber.Clear . . . 863
25.12.6 TJSONIntegerNumber.Clone . . . 863
25.13TJSONNull . . . 863
25.13.1 Description . . . 863
25.13.2 Method overview . . . 864

25.13.3 TJSONNull.JSONType . . . 864
25.13.4 TJSONNull.Clear . . . 864
25.13.5 TJSONNull.Clone . . . 864
25.14 TJSONNumber . . . 864
25.14.1 Description . . . 864
25.14.2 Method overview . . . 865
25.14.3 TJSONNumber.JSONType . . . 865
25.14.4 TJSONNumber.NumberType . . . 865
25.15 TJSONObject . . . 865
25.15.1 Description . . . 865
25.15.2 Method overview . . . 866
25.15.3 Property overview . . . 866
25.15.4 TJSONObject.Create . . . 866
25.15.5 TJSONObject.Destroy . . . 867
25.15.6 TJSONObject.JSONType . . . 867
25.15.7 TJSONObject.Clone . . . 867
25.15.8 TJSONObject.GetEnumerator . . . 868
25.15.9 TJSONObject.Iterate . . . 868
25.15.10 TJSONObject.IndexOf . . . 868
25.15.11 TJSONObject.IndexOfName . . . 868
25.15.12 TJSONObject.Find . . . 869
25.15.13 TJSONObject.Get . . . 869
25.15.14 TJSONObject.Clear . . . 870
25.15.15 TJSONObject.Add . . . 870
25.15.16 TJSONObject.Delete . . . 870
25.15.17 TJSONObject.Remove . . . 871
25.15.18 TJSONObject.Extract . . . 871
25.15.19 TJSONObject.UnquotedMemberNames . . . 871
25.15.20 TJSONObject.Names . . . 872
25.15.21 TJSONObject.Elements . . . 872
25.15.22 TJSONObject.Types . . . 872
25.15.23 TJSONObject.Nulls . . . 873
25.15.24 TJSONObject.Floats . . . 873
25.15.25 TJSONObject.Integers . . . 873
25.15.26 TJSONObject.Int64s . . . 873
25.15.27 TJSONObject.QWords . . . 874
25.15.28 TJSONObject.LargeInts . . . 874
25.15.29 TJSONObject.UnicodeStrings . . . 874
25.15.30 TJSONObject.Strings . . . 875
25.15.31 TJSONObject.Booleans . . . 875

25.15.32TJSONObject.Arrays . . . 875
25.15.33TJSONObject.Objects . . . 876
25.16TJSONQWordNumber . . . 876
25.16.1 Description . . . 876
25.16.2 Method overview . . . 876
25.16.3 TJSONQWordNumber.Create . . . 876
25.16.4 TJSONQWordNumber.NumberType . . . 876
25.16.5 TJSONQWordNumber.Clear . . . 877
25.16.6 TJSONQWordNumber.Clone . . . 877
25.17TJSONString . . . 877
25.17.1 Description . . . 877
25.17.2 Method overview . . . 877
25.17.3 TJSONString.Create . . . 877
25.17.4 TJSONString.JSONType . . . 878
25.17.5 TJSONString.Clear . . . 878
25.17.6 TJSONString.Clone . . . 878

26 Reference for unit 'fpjsonapply' 879
26.1 Used units . . . 879
26.2 Overview . . . 879
26.3 Constants, types and variables . . . 879
26.3.1 Types . . . 879
26.4 TJSONApplier . . . 880
26.4.1 Description . . . 880
26.4.2 Method overview . . . 880
26.4.3 Property overview . . . 881
26.4.4 TJSONApplier.destroy . . . 881
26.4.5 TJSONApplier.Execute . . . 881
26.4.6 TJSONApplier.SourceJSON . . . 882
26.4.7 TJSONApplier.ApplyJSON . . . 882
26.4.8 TJSONApplier.DestJSON . . . 882
26.4.9 TJSONApplier.OwnsJSON . . . 883
26.4.10 TJSONApplier.SourceFileName . . . 883
26.4.11 TJSONApplier.SourcePath . . . 883
26.4.12 TJSONApplier.ApplyFileName . . . 884
26.4.13 TJSONApplier.ApplyPath . . . 884
26.4.14 TJSONApplier.DestFileName . . . 884
26.4.15 TJSONApplier.CloneSource . . . 884
26.4.16 TJSONApplier.CaseInsensitive . . . 885
26.4.17 TJSONApplier.ForceCorrectType . . . 885

26.4.18 TJSONApplier.RemoveNonExisting . . . 885
26.4.19 TJSONApplier.Formatted . . . 886

27 Reference for unit 'fpjsontopas' 887
27.1 Used units . . . 887
27.2 Overview . . . 887
27.3 Using fpjsontopas to generate Pascal classes from JSON . . . 887
27.4 Constants, types and variables . . . 889
27.4.1 Types . . . 889
27.5 EJSONToPascal . . . 889
27.5.1 Description . . . 889
27.6 TJSONToPascal . . . 889
27.6.1 Description . . . 889
27.6.2 Method overview . . . 890
27.6.3 Property overview . . . 890
27.6.4 TJSONToPascal.Create . . . 890
27.6.5 TJSONToPascal.Destroy . . . 890
27.6.6 TJSONToPascal.Execute . . . 891
27.6.7 TJSONToPascal.JSONData . . . 891
27.6.8 TJSONToPascal.JSONStream . . . 891
27.6.9 TJSONToPascal.Active . . . 891
27.6.10 TJSONToPascal.Options . . . 892
27.6.11 TJSONToPascal.JSON . . . 892
27.6.12 TJSONToPascal.Code . . . 892
27.6.13 TJSONToPascal.PropertyMap . . . 892
27.6.14 TJSONToPascal.DestUnitName . . . 893
27.6.15 TJSONToPascal.DefaultParentName . . . 893
27.6.16 TJSONToPascal.IndentSize . . . 893
27.6.17 TJSONToPascal.ExtraUnitNames . . . 893
27.6.18 TJSONToPascal.PropertyTypeSuffix . . . 894
27.6.19 TJSONToPascal.FieldPrefix . . . 894
27.6.20 TJSONToPascal.ObjectConstructorArguments . . . 894
27.6.21 TJSONToPascal.ToplevelObjectClassName . . . 894
27.7 TPropertyMap . . . 895
27.7.1 Description . . . 895
27.7.2 Method overview . . . 895
27.7.3 Property overview . . . 895
27.7.4 TPropertyMap.AddPath . . . 895
27.7.5 TPropertyMap.IndexOfPath . . . 895
27.7.6 TPropertyMap.FindPath . . . 895

27.7.7 TPropertyMap.Map . . . 896
27.8 TPropertyMapItem . . . 896
27.8.1 Description . . . 896
27.8.2 Method overview . . . 896
27.8.3 Property overview . . . 896
27.8.4 TPropertyMapItem.Assign . . . 896
27.8.5 TPropertyMapItem.Generated . . . 897
27.8.6 TPropertyMapItem.Path . . . 897
27.8.7 TPropertyMapItem.TypeName . . . 897
27.8.8 TPropertyMapItem.ParentTypeName . . . 897
27.8.9 TPropertyMapItem.PropertyName . . . 898
27.8.10 TPropertyMapItem.JSONType . . . 898
27.8.11 TPropertyMapItem.SkipType . . . 898

28 Reference for unit 'fpmimetypes' 899
28.1 Used units . . . 899
28.2 Overview . . . 899
28.3 Procedures and functions . . . 899
28.3.1 MimeTypes . . . 899
28.4 TFPMimeTypes . . . 900
28.4.1 Description . . . 900
28.4.2 Method overview . . . 900
28.4.3 TFPMimeTypes.Create . . . 900
28.4.4 TFPMimeTypes.Destroy . . . 900
28.4.5 TFPMimeTypes.Clear . . . 901
28.4.6 TFPMimeTypes.LoadKnownTypes . . . 901
28.4.7 TFPMimeTypes.GetNextExtension . . . 901
28.4.8 TFPMimeTypes.LoadFromStream . . . 901
28.4.9 TFPMimeTypes.LoadFromFile . . . 902
28.4.10 TFPMimeTypes.AddType . . . 902
28.4.11 TFPMimeTypes.GetMimeExtensions . . . 902
28.4.12 TFPMimeTypes.GetMimeType . . . 902
28.4.13 TFPMimeTypes.GetKnownMimeTypes . . . 903
28.4.14 TFPMimeTypes.GetKnownExtensions . . . 903
28.5 TMimeType . . . 903
28.5.1 Description . . . 903
28.5.2 Method overview . . . 903
28.5.3 Property overview . . . 903
28.5.4 TMimeType.Create . . . 904
28.5.5 TMimeType.MergeExtensions . . . 904

28.5.6 TMimeType.MimeType . . . 904
28.5.7 TMimeType.Extensions . . . 904

29 Reference for unit 'fpmkunit' 905
29.1 Used units . . . 905
29.2 Overview . . . 905
29.3 Usage description . . . 905
29.4 Commandline options . . . 908
29.5 Handling dependencies . . . 910
29.6 Constants, types and variables . . . 911
29.6.1 Constants . . . 911
29.6.2 Types . . . 917
29.6.3 Variables . . . 932
29.7 Procedures and functions . . . 933
29.7.1 AddCustomFpmakeCommandlineOption . . . 933
29.7.2 AddLibraryExtension . . . 933
29.7.3 AddProgramExtension . . . 933
29.7.4 ChangeDir . . . 934
29.7.5 CPUSToString . . . 934
29.7.6 CPUToString . . . 934
29.7.7 CurrentCPU . . . 934
29.7.8 CurrentOS . . . 935
29.7.9 FixPath . . . 935
29.7.10 GetCustomFpmakeCommandlineOptionValue . . . 935
29.7.11 GetDefaultLibGCCDir . . . 935
29.7.12 GetImportLibraryFilename . . . 936
29.7.13 GetPluginManager . . . 936
29.7.14 Installer . . . 936
29.7.15 IsDifferentFromBuild . . . 937
29.7.16 IsRelativePath . . . 937
29.7.17 MakeTargetString . . . 937
29.7.18 ModeToString . . . 937
29.7.19 OSesToString . . . 938
29.7.20 OSToString . . . 938
29.7.21 SearchFiles . . . 938
29.7.22 SplitCommand . . . 938
29.7.23 StringToCPU . . . 939
29.7.24 StringToCPUOS . . . 939
29.7.25 StringToCPUS . . . 939
29.7.26 StringToMode . . . 939

29.7.27 StringToOS . . . 940
29.8 ECollectionError . . . 940
29.8.1 Description . . . 940
29.9 EDictionaryError . . . 940
29.9.1 Description . . . 940
29.10EInstallerError . . . 940
29.10.1 Description . . . 940
29.11TBasicDefaults . . . 940
29.11.1 Description . . . 940
29.12TBasicInstaller . . . 940
29.12.1 Description . . . 940
29.12.2 Method overview . . . 941
29.12.3 TBasicInstaller.Create . . . 941
29.13TBuildEngine . . . 941
29.13.1 Description . . . 941
29.13.2 Method overview . . . 942
29.13.3 Property overview . . . 943
29.13.4 TBuildEngine.Create . . . 943
29.13.5 TBuildEngine.Destroy . . . 943
29.13.6 TBuildEngine.AddPathPrefix . . . 943
29.13.7 TBuildEngine.ResolveFileNames . . . 943
29.13.8 TBuildEngine.ClearResolvedFileNames . . . 944
29.13.9 TBuildEngine.ExecuteCommand . . . 944
29.13.10TBuildEngine.CmdCopyFiles . . . 944
29.13.11TBuildEngine.CmdCreateDir . . . 944
29.13.12TBuildEngine.CmdMoveFiles . . . 945
29.13.13TBuildEngine.CmdDeleteFiles . . . 945
29.13.14TBuildEngine.CmdDeleteDestFiles . . . 945
29.13.15TBuildEngine.CmdArchiveFiles . . . 945
29.13.16TBuildEngine.CmdRenameFile . . . 946
29.13.17TBuildEngine.CmdRemoveDirs . . . 946
29.13.18TBuildEngine.CmdRemoveTrees . . . 946
29.13.19TBuildEngine.ExecuteCommands . . . 946
29.13.20TBuildEngine.DependencyOK . . . 946
29.13.21TBuildEngine.GetCompilerCommand . . . 947
29.13.22TBuildEngine.TargetOK . . . 947
29.13.23TBuildEngine.TargetInstallOK . . . 947
29.13.24TBuildEngine.NeedsCompile . . . 947
29.13.25TBuildEngine.Compile . . . 948
29.13.26TBuildEngine.MaybeCompile . . . 948

29.13.27TBuildEngine.CompileDependencies . . . 948
29.13.28TBuildEngine.PackageOK . . . 948
29.13.29TBuildEngine.DoBeforeCompile . . . 949
29.13.30TBuildEngine.DoAfterCompile . . . 949
29.13.31TBuildEngine.DoBeforeInstall . . . 949
29.13.32TBuildEngine.DoAfterInstall . . . 949
29.13.33TBuildEngine.DoBeforeArchive . . . 950
29.13.34TBuildEngine.DoAfterArchive . . . 950
29.13.35TBuildEngine.DoBeforeClean . . . 950
29.13.36TBuildEngine.DoAfterClean . . . 950
29.13.37TBuildEngine.ReadyToCompile . . . 951
29.13.38TBuildEngine.Install . . . 951
29.13.39TBuildEngine.UnInstall . . . 951
29.13.40TBuildEngine.Archive . . . 952
29.13.41TBuildEngine.PkgList . . . 952
29.13.42TBuildEngine.Clean . . . 952
29.13.43TBuildEngine.CheckDependencies . . . 952
29.13.44TBuildEngine.CheckExternalPackage . . . 953
29.13.45TBuildEngine.CreateOutputDir . . . 953
29.13.46TBuildEngine.ZipInstall . . . 953
29.13.47TBuildEngine.Manifest . . . 954
29.13.48TBuildEngine.FPDocProject . . . 954
29.13.49TBuildEngine.Log . . . 954
29.13.50TBuildEngine.Verbose . . . 954
29.13.51TBuildEngine.Interactive . . . 955
29.13.52TBuildEngine.ListMode . . . 955
29.13.53TBuildEngine.ForceCompile . . . 955
29.13.54TBuildEngine.ExternalPackages . . . 955
29.13.55TBuildEngine.StartDir . . . 956
29.13.56TBuildEngine.NotifyEventCollection . . . 956
29.13.57TBuildEngine.OnLog . . . 956
29.14TCommand . . . 956
29.14.1 Description . . . 956
29.14.2 Method overview . . . 957
29.14.3 Property overview . . . 957
29.14.4 TCommand.Destroy . . . 957
29.14.5 TCommand.HaveOptions . . . 957
29.14.6 TCommand.CmdLineOptions . . . 957
29.14.7 TCommand.ParseOptions . . . 958
29.14.8 TCommand.SourceFile . . . 958

29.14.9 TCommand.DestFile . . . 958
29.14.10TCommand.Command . . . 958
29.14.11TCommand.Options . . . 958
29.14.12TCommand.At . . . 959
29.14.13TCommand.IgnoreResult . . . 959
29.14.14TCommand.BeforeCommand . . . 959
29.14.15TCommand.AfterCommand . . . 959
29.15TCommandEnumerator . . . 959
29.15.1 Description . . . 959
29.15.2 Method overview . . . 960
29.15.3 Property overview . . . 960
29.15.4 TCommandEnumerator.GetCurrent . . . 960
29.15.5 TCommandEnumerator.Current . . . 960
29.16TCommands . . . 960
29.16.1 Description . . . 960
29.16.2 Method overview . . . 960
29.16.3 Property overview . . . 961
29.16.4 TCommands.AddCommand . . . 961
29.16.5 TCommands.GetEnumerator . . . 961
29.16.6 TCommands.CommandItems . . . 961
29.16.7 TCommands.Commands . . . 962
29.16.8 TCommands.DefaultAt . . . 962
29.17TCompileWorkerThread . . . 962
29.17.1 Description . . . 962
29.17.2 Method overview . . . 962
29.17.3 TCompileWorkerThread.Create . . . 962
29.17.4 TCompileWorkerThread.Destroy . . . 963
29.18TConditionalDestString . . . 963
29.18.1 Description . . . 963
29.18.2 Property overview . . . 963
29.18.3 TConditionalDestString.DestPath . . . 963
29.19TConditionalDestStrings . . . 963
29.19.1 Description . . . 963
29.19.2 Method overview . . . 964
29.19.3 Property overview . . . 964
29.19.4 TConditionalDestStrings.Create . . . 964
29.19.5 TConditionalDestStrings.Add . . . 964
29.19.6 TConditionalDestStrings.ConditionalStrings . . . 964
29.20TConditionalString . . . 965
29.20.1 Description . . . 965

29.20.2 Method overview . . . 965
29.20.3 Property overview . . . 965
29.20.4 TConditionalString.Create . . . 965
29.20.5 TConditionalString.Value . . . 965
29.20.6 TConditionalString.OSes . . . 966
29.20.7 TConditionalString.CPUs . . . 966
29.21 TConditionalStrings . . . 966
29.21.1 Description . . . 966
29.21.2 Method overview . . . 966
29.21.3 Property overview . . . 966
29.21.4 TConditionalStrings.Create . . . 967
29.21.5 TConditionalStrings.Add . . . 967
29.21.6 TConditionalStrings.ConditionalStrings . . . 967
29.22 TCustomDefaults . . . 967
29.22.1 Description . . . 967
29.22.2 Method overview . . . 968
29.22.3 Property overview . . . 969
29.22.4 TCustomDefaults.Create . . . 969
29.22.5 TCustomDefaults.Destroy . . . 970
29.22.6 TCustomDefaults.InitDefaults . . . 970
29.22.7 TCustomDefaults.HaveOptions . . . 970
29.22.8 TCustomDefaults.IsBuildDifferentFromTarget . . . 970
29.22.9 TCustomDefaults.CompilerDefaults . . . 971
29.22.10 TCustomDefaults.LocalInit . . . 971
29.22.11 TCustomDefaults.LoadFromFile . . . 971
29.22.12 TCustomDefaults.SaveToFile . . . 971
29.22.13 TCustomDefaults.SaveToStream . . . 971
29.22.14 TCustomDefaults.LoadFromStream . . . 972
29.22.15 TCustomDefaults.Target . . . 972
29.22.16 TCustomDefaults.OS . . . 972
29.22.17 TCustomDefaults.CPU . . . 972
29.22.18 TCustomDefaults.CompilerVersion . . . 973
29.22.19 TCustomDefaults.ExplicitOSNone . . . 973
29.22.20 TCustomDefaults.BuildString . . . 973
29.22.21 TCustomDefaults.BuildOS . . . 973
29.22.22 TCustomDefaults.BuildCPU . . . 974
29.22.23 TCustomDefaults.Mode . . . 974
29.22.24 TCustomDefaults.UnixPaths . . . 974
29.22.25 TCustomDefaults.Options . . . 974
29.22.26 TCustomDefaults.NoFPCCfg . . . 975

29.22.27TCustomDefaults.ThreadsAmount . . . 975
29.22.28TCustomDefaults.LocalUnitDir . . . 975
29.22.29TCustomDefaults.GlobalUnitDir . . . 975
29.22.30TCustomDefaults.SearchPath . . . 976
29.22.31TCustomDefaults.Prefix . . . 976
29.22.32TCustomDefaults.ZipPrefix . . . 976
29.22.33TCustomDefaults.BaseInstallDir . . . 976
29.22.34TCustomDefaults.UnitInstallDir . . . 977
29.22.35TCustomDefaults.UnitConfigFilesInstallDir . . . 977
29.22.36TCustomDefaults.BinInstallDir . . . 977
29.22.37TCustomDefaults.LibInstallDir . . . 977
29.22.38TCustomDefaults.DocInstallDir . . . 978
29.22.39TCustomDefaults.ExamplesInstallDir . . . 978
29.22.40TCustomDefaults.FPDocOutputDir . . . 978
29.22.41TCustomDefaults.FPDocOptions . . . 978
29.22.42TCustomDefaults.FPUnitSourcePath . . . 979
29.22.43TCustomDefaults.Compiler . . . 979
29.22.44TCustomDefaults.Copy . . . 979
29.22.45TCustomDefaults.Move . . . 979
29.22.46TCustomDefaults.Remove . . . 980
29.22.47TCustomDefaults.RemoveDir . . . 980
29.22.48TCustomDefaults.RemoveTree . . . 980
29.22.49TCustomDefaults.MkDir . . . 980
29.22.50TCustomDefaults.Archive . . . 981
29.22.51TCustomDefaults.UseEnvironment . . . 981
29.22.52TCustomDefaults.IgnoreInvalidOptions . . . 981
29.22.53TCustomDefaults.BuildMode . . . 981
29.22.54TCustomDefaults.SingleFPDocFile . . . 982
29.22.55TCustomDefaults.InstallExamples . . . 982
29.22.56TCustomDefaults.SkipCrossPrograms . . . 982
29.22.57TCustomDefaults.SkipAllPrograms . . . 982
29.23TCustomInstaller . . . 983
29.23.1 Description . . . 983
29.23.2 Method overview . . . 983
29.23.3 Property overview . . . 983
29.23.4 TCustomInstaller.Create . . . 983
29.23.5 TCustomInstaller.destroy . . . 984
29.23.6 TCustomInstaller.AddPackage . . . 984
29.23.7 TCustomInstaller.AddPackageVariant . . . 984
29.23.8 TCustomInstaller.Run . . . 984

29.23.9 TCustomInstaller.FPMakeOptionsString . . . 985
29.23.10TCustomInstaller.BuildEngine . . . 985
29.23.11TCustomInstaller.Packages . . . 985
29.23.12TCustomInstaller.RunMode . . . 985
29.23.13TCustomInstaller.ListMode . . . 986
29.23.14TCustomInstaller.NotifyEventCollection . . . 986
29.24TDependencies . . . 986
29.24.1 Description . . . 986
29.24.2 Method overview . . . 986
29.24.3 Property overview . . . 986
29.24.4 TDependencies.GetDependency . . . 987
29.24.5 TDependencies.SetDependency . . . 987
29.24.6 TDependencies.Add . . . 987
29.24.7 TDependencies.AddUnit . . . 987
29.24.8 TDependencies.AddInclude . . . 988
29.24.9 TDependencies.Dependencies . . . 988
29.25TDependency . . . 988
29.25.1 Description . . . 988
29.25.2 Method overview . . . 988
29.25.3 Property overview . . . 989
29.25.4 TDependency.Create . . . 989
29.25.5 TDependency.Destroy . . . 989
29.25.6 TDependency.Target . . . 989
29.25.7 TDependency.DependencyType . . . 989
29.25.8 TDependency.TargetFileName . . . 990
29.25.9 TDependency.Version . . . 990
29.25.10TDependency.RequireChecksum . . . 990
29.26TDictionary . . . 990
29.26.1 Description . . . 990
29.26.2 Method overview . . . 991
29.26.3 TDictionary.Create . . . 991
29.26.4 TDictionary.Destroy . . . 991
29.26.5 TDictionary.AddVariable . . . 991
29.26.6 TDictionary.AddFunction . . . 992
29.26.7 TDictionary.RemoveItem . . . 992
29.26.8 TDictionary.GetValue . . . 992
29.26.9 TDictionary.ReplaceStrings . . . 992
29.26.10TDictionary.Substitute . . . 993
29.27TFPCDefaults . . . 993
29.27.1 Description . . . 993

29.27.2 Method overview . . . 993
29.27.3 TFPCDefaults.CompilerDefaults . . . 993
29.28TFPCInstaller . . . 994
29.28.1 Description . . . 994
29.28.2 Method overview . . . 994
29.28.3 TFPCInstaller.Create . . . 994
29.29TfpmPlugin . . . 994
29.29.1 Description . . . 994
29.29.2 Method overview . . . 994
29.29.3 Property overview . . . 994
29.29.4 TfpmPlugin.BeforeResolvePackagePath . . . 995
29.29.5 TfpmPlugin.ResolvePackagePath . . . 995
29.29.6 TfpmPlugin.AfterResolvePackagePath . . . 995
29.29.7 TfpmPlugin.Name . . . 995
29.30TfpmPluginManager . . . 996
29.30.1 Description . . . 996
29.30.2 Method overview . . . 996
29.30.3 TfpmPluginManager.Destroy . . . 996
29.30.4 TfpmPluginManager.RegisterPlugin . . . 996
29.30.5 TfpmPluginManager.BeforeResolvePackagePath . . . 996
29.30.6 TfpmPluginManager.ResolvePackagePath . . . 997
29.30.7 TfpmPluginManager.AfterResolvePackagePath . . . 997
29.31TfpmResolvePackagePathsPlugin . . . 997
29.31.1 Description . . . 997
29.31.2 Method overview . . . 997
29.31.3 TfpmResolvePackagePathsPlugin.BeforeResolvePackagePath . . . 998
29.31.4 TfpmResolvePackagePathsPlugin.ResolvePackagePath . . . 998
29.32TFPVersion . . . 998
29.32.1 Description . . . 998
29.32.2 Method overview . . . 998
29.32.3 Property overview . . . 998
29.32.4 TFPVersion.Clear . . . 999
29.32.5 TFPVersion.Assign . . . 999
29.32.6 TFPVersion.CompareVersion . . . 999
29.32.7 TFPVersion.SameVersion . . . 999
29.32.8 TFPVersion.AsString . . . 1000
29.32.9 TFPVersion.Empty . . . 1000
29.32.10TFPVersion.Major . . . 1000
29.32.11TFPVersion.Minor . . . 1000
29.32.12TFPVersion.Micro . . . 1001

29.32.13TFPVersion.Build 1001
29.33TFunctionItem 1001
29.33.1 Description 1001
29.33.2 Method overview 1002
29.33.3 TFunctionItem.Create 1002
29.34TNamedCollection 1002
29.34.1 Description 1002
29.34.2 Method overview 1002
29.34.3 Property overview 1002
29.34.4 TNamedCollection.IndexOfName 1002
29.34.5 TNamedCollection.ItemByName 1002
29.34.6 TNamedCollection.UniqueNames 1003
29.34.7 TNamedCollection.Items 1003
29.35TNamedItem 1003
29.35.1 Description 1003
29.35.2 Property overview 1003
29.35.3 TNamedItem.Name 1003
29.36TNamedItemList 1004
29.36.1 Description 1004
29.36.2 Method overview 1004
29.36.3 Property overview 1004
29.36.4 TNamedItemList.IndexOfName 1004
29.36.5 TNamedItemList.ItemByName 1004
29.36.6 TNamedItemList.NamedItems 1004
29.37TNotifyEventCollection 1005
29.37.1 Description 1005
29.37.2 Method overview 1005
29.37.3 TNotifyEventCollection.create 1005
29.37.4 TNotifyEventCollection.GetEnumerator 1005
29.37.5 TNotifyEventCollection.AppendEvent 1005
29.37.6 TNotifyEventCollection.AppendProcEvent 1006
29.37.7 TNotifyEventCollection.CallEvents 1006
29.38TNotifyEventEnumerator 1006
29.38.1 Description 1006
29.38.2 Method overview 1006
29.38.3 Property overview 1006
29.38.4 TNotifyEventEnumerator.GetCurrent 1007
29.38.5 TNotifyEventEnumerator.Current 1007
29.39TNotifyEventItem 1007
29.39.1 Description 1007

29.39.2 Method overview . . . 1007
29.39.3 Property overview . . . 1007
29.39.4 TNotifyEventItem.CallEvent . . . 1008
29.39.5 TNotifyEventItem.OnAction . . . 1008
29.39.6 TNotifyEventItem.OnEvent . . . 1008
29.39.7 TNotifyEventItem.OnProcEvent . . . 1008
29.40TPackage . . . 1009
29.40.1 Description . . . 1009
29.40.2 Method overview . . . 1010
29.40.3 Property overview . . . 1012
29.40.4 TPackage.Create . . . 1013
29.40.5 TPackage.destroy . . . 1013
29.40.6 TPackage.HaveOptions . . . 1013
29.40.7 TPackage.GetUnitsOutputDir . . . 1013
29.40.8 TPackage.GetUnitConfigOutputFilename . . . 1014
29.40.9 TPackage.InheritPackageVariantsFromDependency . . . 1014
29.40.10TPackage.GetPackageVariantsByName . . . 1014
29.40.11TPackage.SetUnitsOutputDir . . . 1014
29.40.12TPackage.GetPackageUnitInstallDir . . . 1014
29.40.13TPackage.SetPackageUnitInstallDir . . . 1015
29.40.14TPackage.GetBinOutputDir . . . 1015
29.40.15TPackage.GetCleanFiles . . . 1015
29.40.16TPackage.GetInstallFiles . . . 1015
29.40.17TPackage.GetInstallSourceFiles . . . 1015
29.40.18TPackage.GetArchiveFiles . . . 1016
29.40.19TPackage.GetArchiveSourceFiles . . . 1016
29.40.20TPackage.GetManifest . . . 1016
29.40.21TPackage.ListPackage . . . 1016
29.40.22TPackage.AddPackageVariant . . . 1017
29.40.23TPackage.ApplyPackageVariantToCompilerOptions . . . 1017
29.40.24TPackage.SetDefaultPackageVariant . . . 1017
29.40.25TPackage.LoadUnitConfigFromFile . . . 1017
29.40.26TPackage.SaveUnitConfigToFile . . . 1017
29.40.27TPackage.EnterResolveDirsCS . . . 1018
29.40.28TPackage.LeaveResolveDirsCS . . . 1018
29.40.29TPackage.Version . . . 1018
29.40.30TPackage.FileName . . . 1018
29.40.31TPackage.ShortName . . . 1019
29.40.32TPackage.HomepageURL . . . 1019
29.40.33TPackage.DownloadURL . . . 1019

29.40.34 TPackage.Email 1019
29.40.35 TPackage.Author 1020
29.40.36 TPackage.License 1020
29.40.37 TPackage.Directory 1020
29.40.38 TPackage.Description 1020
29.40.39 TPackage.DescriptionFile 1021
29.40.40 TPackage.InstalledChecksum 1021
29.40.41 TPackage.IsFPMakeAddIn 1021
29.40.42 TPackage.SupportBuildModes 1021
29.40.43 TPackage.BuildMode 1022
29.40.44 TPackage.Flags 1022
29.40.45 TPackage.PackageVersion 1022
29.40.46 TPackage.TransmitOptions 1022
29.40.47 TPackage.OSes 1023
29.40.48 TPackage.CPUs 1023
29.40.49 TPackage.NeedLibC 1023
29.40.50 TPackage.Options 1023
29.40.51 TPackage.UnitPath 1024
29.40.52 TPackage.ObjectPath 1024
29.40.53 TPackage.IncludePath 1024
29.40.54 TPackage.SourcePath 1024
29.40.55 TPackage.ExamplePath 1025
29.40.56 TPackage.TestPath 1025
29.40.57 TPackage.FPDocFormat 1025
29.40.58 TPackage.InstallFiles 1025
29.40.59 TPackage.CleanFiles 1026
29.40.60 TPackage.Dependencies 1026
29.40.61 TPackage.Commands 1026
29.40.62 TPackage.State 1026
29.40.63 TPackage.Targets 1027
29.40.64 TPackage.Sources 1027
29.40.65 TPackage.UnitDir 1027
29.40.66 TPackage.UnitConfigFileName 1027
29.40.67 TPackage.BeforeCompile 1028
29.40.68 TPackage.BeforeCompileProc 1028
29.40.69 TPackage.AfterCompile 1028
29.40.70 TPackage.AfterCompileProc 1028
29.40.71 TPackage.BeforeInstall 1029
29.40.72 TPackage.BeforeInstallProc 1029
29.40.73 TPackage.AfterInstall 1029

29.40.74TPackage.AfterInstallProc . . . 1029
29.40.75TPackage.BeforeClean . . . 1030
29.40.76TPackage.BeforeCleanProc . . . 1030
29.40.77TPackage.AfterClean . . . 1030
29.40.78TPackage.AfterCleanProc . . . 1030
29.40.79TPackage.BeforeArchive . . . 1031
29.40.80TPackage.BeforeArchiveProc . . . 1031
29.40.81TPackage.AfterArchive . . . 1031
29.40.82TPackage.AfterArchiveProc . . . 1031
29.40.83TPackage.BeforeManifest . . . 1032
29.40.84TPackage.BeforeManifestProc . . . 1032
29.40.85TPackage.AfterManifest . . . 1032
29.40.86TPackage.AfterManifestProc . . . 1032
29.40.87TPackage.BeforePkgList . . . 1033
29.40.88TPackage.BeforePkgListProc . . . 1033
29.40.89TPackage.AfterPkgList . . . 1033
29.40.90TPackage.AfterPkgListProc . . . 1033
29.41TPackageDictionary . . . 1034
29.41.1 Description . . . 1034
29.41.2 Method overview . . . 1034
29.41.3 Property overview . . . 1034
29.41.4 TPackageDictionary.GetValue . . . 1034
29.41.5 TPackageDictionary.MasterDictionary . . . 1034
29.42TPackageEnumerator . . . 1035
29.42.1 Description . . . 1035
29.42.2 Method overview . . . 1035
29.42.3 Property overview . . . 1035
29.42.4 TPackageEnumerator.GetCurrent . . . 1035
29.42.5 TPackageEnumerator.Current . . . 1035
29.43TPackages . . . 1035
29.43.1 Description . . . 1035
29.43.2 Method overview . . . 1036
29.43.3 Property overview . . . 1036
29.43.4 TPackages.AddPackage . . . 1036
29.43.5 TPackages.GetEnumerator . . . 1036
29.43.6 TPackages.Packages . . . 1036
29.43.7 TPackages.PackageItems . . . 1037
29.44TPackageVariant . . . 1037
29.44.1 Description . . . 1037
29.44.2 Method overview . . . 1037

29.44.3 Property overview . . . 1037
29.44.4 TPackageVariant.Create . . . 1037
29.44.5 TPackageVariant.Destroy . . . 1037
29.44.6 TPackageVariant.Options . . . 1038
29.44.7 TPackageVariant.IncludePath . . . 1038
29.44.8 TPackageVariant.SourcePath . . . 1038
29.44.9 TPackageVariant.Targets . . . 1038
29.45TPackageVariantEnumerator . . . 1039
29.45.1 Description . . . 1039
29.45.2 Method overview . . . 1039
29.45.3 Property overview . . . 1039
29.45.4 TPackageVariantEnumerator.GetCurrent . . . 1039
29.45.5 TPackageVariantEnumerator.Current . . . 1039
29.46TPackageVariants . . . 1039
29.46.1 Description . . . 1039
29.46.2 Method overview . . . 1040
29.46.3 Property overview . . . 1040
29.46.4 TPackageVariants.Add . . . 1040
29.46.5 TPackageVariants.GetEnumerator . . . 1040
29.46.6 TPackageVariants.Name . . . 1040
29.46.7 TPackageVariants.MasterPackage . . . 1041
29.46.8 TPackageVariants.DefaultPackageVariant . . . 1041
29.46.9 TPackageVariants.ActivePackageVariant . . . 1041
29.46.10TPackageVariants.DefaultPackageVariantName . . . 1041
29.46.11TPackageVariants.ActivePackageVariantName . . . 1042
29.46.12TPackageVariants.IsInheritable . . . 1042
29.46.13TPackageVariants.AutoAddToPackage . . . 1042
29.47TResourceFile . . . 1042
29.47.1 Description . . . 1042
29.48TResourceFiles . . . 1043
29.48.1 Description . . . 1043
29.48.2 Method overview . . . 1043
29.48.3 TResourceFiles.GetInstallFiles . . . 1043
29.49TSource . . . 1043
29.49.1 Description . . . 1043
29.49.2 Method overview . . . 1043
29.49.3 Property overview . . . 1043
29.49.4 TSource.Create . . . 1044
29.49.5 TSource.Destroy . . . 1044
29.49.6 TSource.GetInstallFiles . . . 1044

29.49.7 TSource.SourceType . . . 1044
29.49.8 TSource.InstallSourcePath . . . 1045
29.50TSourceEnumerator . . . 1045
29.50.1 Description . . . 1045
29.50.2 Method overview . . . 1045
29.50.3 Property overview . . . 1045
29.50.4 TSourceEnumerator.GetCurrent . . . 1045
29.50.5 TSourceEnumerator.Current . . . 1045
29.51TSources . . . 1046
29.51.1 Description . . . 1046
29.51.2 Method overview . . . 1046
29.51.3 Property overview . . . 1046
29.51.4 TSources.AddDoc . . . 1046
29.51.5 TSources.AddSrc . . . 1046
29.51.6 TSources.AddExample . . . 1047
29.51.7 TSources.AddTest . . . 1047
29.51.8 TSources.GetEnumerator . . . 1047
29.51.9 TSources.AddDocFiles . . . 1048
29.51.10TSources.AddSrcFiles . . . 1048
29.51.11TSources.AddExampleFiles . . . 1048
29.51.12TSources.AddTestFiles . . . 1048
29.51.13TSources.SourceItems . . . 1049
29.52TTarget . . . 1049
29.52.1 Description . . . 1049
29.52.2 Method overview . . . 1049
29.52.3 Property overview . . . 1050
29.52.4 TTarget.Create . . . 1050
29.52.5 TTarget.Destroy . . . 1050
29.52.6 TTarget.AssignTo . . . 1051
29.52.7 TTarget.GetOutputFileName . . . 1051
29.52.8 TTarget.HaveOptions . . . 1051
29.52.9 TTarget.SetName . . . 1051
29.52.10TTarget.SetExeName . . . 1051
29.52.11TTarget.SetXML . . . 1052
29.52.12TTarget.GetCleanFiles . . . 1052
29.52.13TTarget.GetInstallFiles . . . 1052
29.52.14TTarget.GetArchiveFiles . . . 1052
29.52.15TTarget.Dependencies . . . 1053
29.52.16TTarget.ResourceFiles . . . 1053
29.52.17TTarget.Commands . . . 1053

29.52.18 TTarget.State . . . 1053
29.52.19 TTarget.TargetType . . . 1054
29.52.20 TTarget.OSes . . . 1054
29.52.21 TTarget.CPUs . . . 1054
29.52.22 TTarget.Mode . . . 1054
29.52.23 TTarget.Options . . . 1055
29.52.24 TTarget.SourceFileName . . . 1055
29.52.25 TTarget.UnitFileName . . . 1055
29.52.26 TTarget.ObjectFileName . . . 1055
29.52.27 TTarget.RSTFileName . . . 1056
29.52.28 TTarget.RSJFileName . . . 1056
29.52.29 TTarget.FPCTarget . . . 1056
29.52.30 TTarget.Extension . . . 1056
29.52.31 TTarget.FileType . . . 1057
29.52.32 TTarget.Directory . . . 1057
29.52.33 TTarget.ResourceStrings . . . 1057
29.52.34 TTarget.Install . . . 1057
29.52.35 TTarget.TargetSourceFileName . . . 1058
29.52.36 TTarget.ObjectPath . . . 1058
29.52.37 TTarget.UnitPath . . . 1058
29.52.38 TTarget.IncludePath . . . 1058
29.52.39 TTarget.XML . . . 1059
29.52.40 TTarget.IsFPMakePlugin . . . 1059
29.52.41 TTarget.BeforeCompile . . . 1059
29.52.42 TTarget.AfterCompile . . . 1059
29.52.43 TTarget.BeforeClean . . . 1060
29.52.44 TTarget.AfterClean . . . 1060
29.53 TTargetEnumerator . . . 1060
29.53.1 Description . . . 1060
29.53.2 Method overview . . . 1060
29.53.3 Property overview . . . 1060
29.53.4 TTargetEnumerator.GetCurrent . . . 1060
29.53.5 TTargetEnumerator.Current . . . 1061
29.54 TTargets . . . 1061
29.54.1 Description . . . 1061
29.54.2 Method overview . . . 1061
29.54.3 Property overview . . . 1061
29.54.4 TTargets.AddFPDoc . . . 1061
29.54.5 TTargets.AddUnit . . . 1062
29.54.6 TTargets.AddImplicitUnit . . . 1062

29.54.7 TTargets.AddProgram . . . 1062
29.54.8 TTargets.AddLibrary . . . 1063
29.54.9 TTargets.AddExampleUnit . . . 1063
29.54.10 TTargets.AddExampleProgram . . . 1063
29.54.11 TTargets.GetEnumerator . . . 1064
29.54.12 TTargets.Targets . . . 1064
29.54.13 TTargets.TargetItems . . . 1064
29.55 TValueItem . . . 1064
29.55.1 Description . . . 1064
29.55.2 Method overview . . . 1065
29.55.3 TValueItem.Create . . . 1065

30 Reference for unit 'fpobserver' 1066
30.1 Used units . . . 1066
30.2 Overview . . . 1066
30.3 Constants, types and variables . . . 1066
30.3.1 Types . . . 1066
30.4 Procedures and functions . . . 1067
30.4.1 MediatorError . . . 1067
30.4.2 MediatorManager . . . 1067
30.5 EMediator . . . 1068
30.5.1 Description . . . 1068
30.6 TBaseListMediator . . . 1068
30.6.1 Description . . . 1068
30.6.2 Method overview . . . 1068
30.6.3 Property overview . . . 1068
30.6.4 TBaseListMediator.Create . . . 1068
30.6.5 TBaseListMediator.Destroy . . . 1068
30.6.6 TBaseListMediator.CompositeMediator . . . 1069
30.6.7 TBaseListMediator.ObservedChanged . . . 1069
30.6.8 TBaseListMediator.SelectedObject . . . 1069
30.6.9 TBaseListMediator.OnBeforeSetupField . . . 1069
30.7 TBaseMediator . . . 1070
30.7.1 Description . . . 1070
30.7.2 Interfaces overview . . . 1070
30.7.3 Method overview . . . 1070
30.7.4 Property overview . . . 1070
30.7.5 TBaseMediator.Destroy . . . 1071
30.7.6 TBaseMediator.ObjectToView . . . 1071
30.7.7 TBaseMediator.ViewToObject . . . 1071

30.7.8 TBaseMediator.ViewClass . . . 1071
30.7.9 TBaseMediator.CompositeMediator . . . 1072
30.7.10 TBaseMediator.Subject . . . 1072
30.7.11 TBaseMediator.View . . . 1072
30.7.12 TBaseMediator.Valuelist . . . 1072
30.7.13 TBaseMediator.SubjectPropertyName . . . 1073
30.7.14 TBaseMediator.Active . . . 1073
30.7.15 TBaseMediator.ReadOnly . . . 1073
30.7.16 TBaseMediator.OnViewToObject . . . 1074
30.7.17 TBaseMediator.OnObjectToView . . . 1074
30.8 TColumnsListItemMediator . . . 1074
30.8.1 Description . . . 1074
30.8.2 Property overview . . . 1074
30.8.3 TColumnsListItemMediator.FieldsInfo . . . 1074
30.9 TColumnsListMediator . . . 1075
30.9.1 Description . . . 1075
30.9.2 Method overview . . . 1075
30.9.3 Property overview . . . 1075
30.9.4 TColumnsListMediator.Create . . . 1075
30.9.5 TColumnsListMediator.Destroy . . . 1075
30.9.6 TColumnsListMediator.DisplayNames . . . 1076
30.9.7 TColumnsListMediator.FieldsInfo . . . 1076
30.10TComponentMediator . . . 1076
30.10.1 Description . . . 1076
30.10.2 Property overview . . . 1076
30.10.3 TComponentMediator.ViewComponent . . . 1077
30.10.4 TComponentMediator.ViewPropertyName . . . 1077
30.11TListItemMediator . . . 1077
30.11.1 Description . . . 1077
30.11.2 Interfaces overview . . . 1077
30.11.3 Method overview . . . 1077
30.11.4 Property overview . . . 1078
30.11.5 TListItemMediator.Destroy . . . 1078
30.11.6 TListItemMediator.MaybeObjectToView . . . 1078
30.11.7 TListItemMediator.OnBeforeSetupField . . . 1078
30.11.8 TListItemMediator.Subject . . . 1078
30.11.9 TListItemMediator.Active . . . 1079
30.12TMediatorDef . . . 1079
30.12.1 Description . . . 1079
30.12.2 Method overview . . . 1079

30.12.3 Property overview . . . 1079
30.12.4 TMediatorDef.Handles . . . 1079
30.12.5 TMediatorDef.BetterMatch . . . 1080
30.12.6 TMediatorDef.MediatorClass . . . 1080
30.12.7 TMediatorDef.MinSubjectClass . . . 1080
30.12.8 TMediatorDef.PropertyTypes . . . 1080
30.12.9 TMediatorDef.PropertyName . . . 1081
30.13TMediatorDefs . . . 1081
30.13.1 Description . . . 1081
30.13.2 Method overview . . . 1081
30.13.3 Property overview . . . 1081
30.13.4 TMediatorDefs.AddDef . . . 1081
30.13.5 TMediatorDefs.Defs . . . 1081
30.14TMediatorFieldInfo . . . 1082
30.14.1 Description . . . 1082
30.14.2 Method overview . . . 1082
30.14.3 Property overview . . . 1082
30.14.4 TMediatorFieldInfo.Assign . . . 1082
30.14.5 TMediatorFieldInfo.AsString . . . 1082
30.14.6 TMediatorFieldInfo.Caption . . . 1083
30.14.7 TMediatorFieldInfo.PropertyName . . . 1083
30.14.8 TMediatorFieldInfo.Width . . . 1083
30.14.9 TMediatorFieldInfo.Alignment . . . 1084
30.15TMediatorFieldInfoList . . . 1084
30.15.1 Description . . . 1084
30.15.2 Method overview . . . 1084
30.15.3 Property overview . . . 1084
30.15.4 TMediatorFieldInfoList.AddFieldInfo . . . 1084
30.15.5 TMediatorFieldInfoList.FieldInfo . . . 1085
30.15.6 TMediatorFieldInfoList.AsString . . . 1085
30.16TMediatorManager . . . 1085
30.16.1 Description . . . 1085
30.16.2 Method overview . . . 1085
30.16.3 Property overview . . . 1086
30.16.4 TMediatorManager.Create . . . 1086
30.16.5 TMediatorManager.Destroy . . . 1086
30.16.6 TMediatorManager.FindDefFor . . . 1086
30.16.7 TMediatorManager.RegisterMediator . . . 1087
30.16.8 TMediatorManager.Defs . . . 1087
30.17TObservedHook . . . 1087

30.17.1 Description . . . 1087
30.17.2 Interfaces overview . . . 1087
30.17.3 Method overview . . . 1088
30.17.4 Property overview . . . 1088
30.17.5 TObservedHook.CreateSender . . . 1088
30.17.6 TObservedHook.Destroy . . . 1088
30.17.7 TObservedHook.FPOAttachObserver . . . 1088
30.17.8 TObservedHook.FPODetachObserver . . . 1089
30.17.9 TObservedHook.Changed . . . 1089
30.17.10TObservedHook.AddItem . . . 1089
30.17.11TObservedHook.DeleteItem . . . 1089
30.17.12TObservedHook.CustomNotify . . . 1090
30.17.13TObservedHook.FPONotifyObservers . . . 1090
30.17.14TObservedHook.Sender . . . 1090

31 Reference for unit 'fpTemplate' 1091
31.1 Used units . . . 1091
31.2 Overview . . . 1091
31.3 Using templates with parameters . . . 1091
31.4 Basic usage of the template parser . . . 1095
31.5 Constants, types and variables . . . 1098
31.5.1 Constants . . . 1098
31.5.2 Types . . . 1098
31.5.3 Variables . . . 1099
31.6 ETemplateParser . . . 1099
31.6.1 Description . . . 1099
31.7 TFPCustomTemplate . . . 1099
31.7.1 Description . . . 1099
31.7.2 Method overview . . . 1100
31.7.3 Property overview . . . 1100
31.7.4 TFPCustomTemplate.HasContent . . . 1100
31.7.5 TFPCustomTemplate.GetContent . . . 1100
31.7.6 TFPCustomTemplate.Assign . . . 1100
31.7.7 TFPCustomTemplate.StartDelimiter . . . 1101
31.7.8 TFPCustomTemplate.EndDelimiter . . . 1101
31.7.9 TFPCustomTemplate.ParamStartDelimiter . . . 1101
31.7.10 TFPCustomTemplate.ParamEndDelimiter . . . 1101
31.7.11 TFPCustomTemplate.ParamValueSeparator . . . 1102
31.7.12 TFPCustomTemplate.FileName . . . 1102
31.7.13 TFPCustomTemplate.Template . . . 1102

31.7.14 TFPCustomTemplate.OnGetParam . . . 1102
31.7.15 TFPCustomTemplate.OnReplaceTag . . . 1103
31.7.16 TFPCustomTemplate.AllowTagParams . . . 1103
31.8 TFPTemplate . . . 1103
31.8.1 Description . . . 1103
31.8.2 Property overview . . . 1103
31.8.3 TFPTemplate.FileName . . . 1104
31.8.4 TFPTemplate.Template . . . 1104
31.8.5 TFPTemplate.AllowTagParams . . . 1104
31.8.6 TFPTemplate.OnReplaceTag . . . 1104
31.8.7 TFPTemplate.StartDelimiter . . . 1105
31.8.8 TFPTemplate.EndDelimiter . . . 1105
31.8.9 TFPTemplate.ParamStartDelimiter . . . 1105
31.8.10 TFPTemplate.ParamEndDelimiter . . . 1105
31.8.11 TFPTemplate.ParamValueSeparator . . . 1106
31.8.12 TFPTemplate.OnGetParam . . . 1106
31.9 TTemplateParser . . . 1106
31.9.1 Description . . . 1106
31.9.2 Method overview . . . 1106
31.9.3 Property overview . . . 1107
31.9.4 TTemplateParser.Create . . . 1107
31.9.5 TTemplateParser.Destroy . . . 1107
31.9.6 TTemplateParser.Clear . . . 1107
31.9.7 TTemplateParser.ReplaceTag . . . 1108
31.9.8 TTemplateParser.GetParam . . . 1108
31.9.9 TTemplateParser.GetTagParams . . . 1108
31.9.10 TTemplateParser.ParseString . . . 1108
31.9.11 TTemplateParser.ParseStream . . . 1109
31.9.12 TTemplateParser.ParseStrings . . . 1109
31.9.13 TTemplateParser.ParseFiles . . . 1109
31.9.14 TTemplateParser.OnGetParam . . . 1109
31.9.15 TTemplateParser.OnReplaceTag . . . 1110
31.9.16 TTemplateParser.StartDelimiter . . . 1110
31.9.17 TTemplateParser.EndDelimiter . . . 1110
31.9.18 TTemplateParser.ParamStartDelimiter . . . 1110
31.9.19 TTemplateParser.ParamEndDelimiter . . . 1111
31.9.20 TTemplateParser.ParamValueSeparator . . . 1111
31.9.21 TTemplateParser.Values . . . 1111
31.9.22 TTemplateParser.ValuesByIndex . . . 1111
31.9.23 TTemplateParser.NamesByIndex . . . 1112

31.9.24 TTemplateParser.ValueCount . . . 1112
31.9.25 TTemplateParser.Recursive . . . 1112
31.9.26 TTemplateParser.AllowTagParams . . . 1112

32 Reference for unit 'fptimer' 1113
32.1 Used units . . . 1113
32.2 Overview . . . 1113
32.3 Constants, types and variables . . . 1113
32.3.1 Types . . . 1113
32.3.2 Variables . . . 1113
32.4 TFPCustomTimer . . . 1114
32.4.1 Description . . . 1114
32.4.2 Method overview . . . 1114
32.4.3 TFPCustomTimer.Create . . . 1114
32.4.4 TFPCustomTimer.Destroy . . . 1114
32.4.5 TFPCustomTimer.StartTimer . . . 1115
32.4.6 TFPCustomTimer.StopTimer . . . 1115
32.5 TFPTimer . . . 1115
32.5.1 Description . . . 1115
32.5.2 Property overview . . . 1115
32.5.3 TFPTimer.Enabled . . . 1115
32.5.4 TFPTimer.Interval . . . 1116
32.5.5 TFPTimer.UseTimerThread . . . 1116
32.5.6 TFPTimer.OnTimer . . . 1116
32.5.7 TFPTimer.OnStartTimer . . . 1116
32.5.8 TFPTimer.OnStopTimer . . . 1117
32.6 TFPTimerDriver . . . 1117
32.6.1 Description . . . 1117
32.6.2 Method overview . . . 1117
32.6.3 Property overview . . . 1117
32.6.4 TFPTimerDriver.Create . . . 1117
32.6.5 TFPTimerDriver.StartTimer . . . 1117
32.6.6 TFPTimerDriver.StopTimer . . . 1118
32.6.7 TFPTimerDriver.Timer . . . 1118
32.6.8 TFPTimerDriver.TimerStarted . . . 1118

33 Reference for unit 'fpWeb' 1119
33.1 Used units . . . 1119
33.2 Overview . . . 1119
33.3 Constants, types and variables . . . 1119
33.3.1 Resource strings . . . 1119

33.3.2 Types . . . 1120
33.4 EFPWebError . . . 1120
33.4.1 Description . . . 1120
33.5 TCustomFPWebModule . . . 1120
33.5.1 Description . . . 1120
33.5.2 Method overview . . . 1120
33.5.3 Property overview . . . 1121
33.5.4 TCustomFPWebModule.CreateNew . . . 1121
33.5.5 TCustomFPWebModule.Destroy . . . 1121
33.5.6 TCustomFPWebModule.HandleRequest . . . 1121
33.5.7 TCustomFPWebModule.Actions . . . 1122
33.5.8 TCustomFPWebModule.ActionVar . . . 1122
33.5.9 TCustomFPWebModule.BeforeRequest . . . 1122
33.5.10 TCustomFPWebModule.OnRequest . . . 1123
33.5.11 TCustomFPWebModule.AfterResponse . . . 1123
33.5.12 TCustomFPWebModule.OnGetAction . . . 1123
33.5.13 TCustomFPWebModule.DefActionWhenUnknown . . . 1123
33.5.14 TCustomFPWebModule.ModuleTemplate . . . 1124
33.5.15 TCustomFPWebModule.OnGetParam . . . 1124
33.5.16 TCustomFPWebModule.OnTemplateContent . . . 1124
33.5.17 TCustomFPWebModule.Request . . . 1124
33.5.18 TCustomFPWebModule.Response . . . 1125
33.6 TFPWebAction . . . 1125
33.6.1 Description . . . 1125
33.6.2 Method overview . . . 1125
33.6.3 Property overview . . . 1125
33.6.4 TFPWebAction.create . . . 1125
33.6.5 TFPWebAction.destroy . . . 1126
33.6.6 TFPWebAction.Assign . . . 1126
33.6.7 TFPWebAction.Contents . . . 1126
33.6.8 TFPWebAction.OnRequest . . . 1126
33.6.9 TFPWebAction.Template . . . 1127
33.7 TFPWebActions . . . 1127
33.7.1 Description . . . 1127
33.7.2 Method overview . . . 1127
33.7.3 Property overview . . . 1127
33.7.4 TFPWebActions.Add . . . 1127
33.7.5 TFPWebActions.ActionByName . . . 1128
33.7.6 TFPWebActions.FindAction . . . 1128
33.7.7 TFPWebActions.FPWebActions . . . 1128

33.7.8 TFPWebActions.ActionVar . . . 1128
33.7.9 TFPWebActions.CurrentAction . . . 1129
33.8 TFPWebModule . . . 1129
33.8.1 Description . . . 1129
33.8.2 Property overview . . . 1129
33.8.3 TFPWebModule.Actions . . . 1129
33.8.4 TFPWebModule.ActionVar . . . 1130
33.8.5 TFPWebModule.BeforeRequest . . . 1130
33.8.6 TFPWebModule.OnRequest . . . 1130
33.8.7 TFPWebModule.AfterResponse . . . 1130
33.8.8 TFPWebModule.OnGetAction . . . 1131
33.8.9 TFPWebModule.DefActionWhenUnknown . . . 1131
33.8.10 TFPWebModule.CreateSession . . . 1131
33.8.11 TFPWebModule.Session . . . 1131
33.8.12 TFPWebModule.Kind . . . 1132
33.8.13 TFPWebModule.OnNewSession . . . 1132
33.8.14 TFPWebModule.OnSessionExpired . . . 1132
33.8.15 TFPWebModule.AfterInitModule . . . 1132
33.8.16 TFPWebModule.CORS . . . 1133
33.9 TTemplateVar . . . 1133
33.9.1 Description . . . 1133
33.9.2 Method overview . . . 1133
33.9.3 Property overview . . . 1133
33.9.4 TTemplateVar.Assign . . . 1133
33.9.5 TTemplateVar.GetDisplayName . . . 1133
33.9.6 TTemplateVar.Name . . . 1134
33.9.7 TTemplateVar.Value . . . 1134
33.10 TTemplateVars . . . 1134
33.10.1 Description . . . 1134
33.10.2 Method overview . . . 1134
33.10.3 Property overview . . . 1134
33.10.4 TTemplateVars.IndexOfVar . . . 1135
33.10.5 TTemplateVars.VarByName . . . 1135
33.10.6 TTemplateVars.FindVar . . . 1135
33.10.7 TTemplateVars.Variables . . . 1135

34 Reference for unit 'gettext' 1136
34.1 Used units . . . 1136
34.2 Overview . . . 1136
34.3 Constants, types and variables . . . 1136

34.3.1 Constants . . . 1136
34.3.2 Types . . . 1136
34.3.3 Variables . . . 1137
34.4 Procedures and functions . . . 1137
34.4.1 GetLanguageIDs . . . 1137
34.4.2 TranslateResourceStrings . . . 1138
34.4.3 TranslateUnitResourceStrings . . . 1138
34.5 TMOFileHeader . . . 1138
34.6 TMOStringInfo . . . 1139
34.7 EMOFileError . . . 1139
34.7.1 Description . . . 1139
34.8 TMOFile . . . 1139
34.8.1 Description . . . 1139
34.8.2 Method overview . . . 1139
34.8.3 TMOFile.Create . . . 1139
34.8.4 TMOFile.Destroy . . . 1140
34.8.5 TMOFile.Translate . . . 1140

35 Reference for unit 'HTTPDefs' 1141
35.1 Used units . . . 1141
35.2 Overview . . . 1141
35.3 Using TRequest and TResponse Classes . . . 1141
35.4 Working with HTTP Cookies . . . 1144
35.5 Understanding TCORSSupport.HandleRequest Return Value . . . 1146
35.6 CORS Support Usage . . . 1148
35.7 Working with HTTP Sessions . . . 1149
35.8 Constants, types and variables . . . 1151
35.8.1 Constants . . . 1151
35.8.2 Types . . . 1158
35.8.3 Variables . . . 1161
35.9 Procedures and functions . . . 1162
35.9.1 HTTPDecode . . . 1162
35.9.2 HTTPEncode . . . 1162
35.9.3 IncludeHTTPPathDelimiter . . . 1162
35.10EHTTP . . . 1162
35.10.1 Description . . . 1162
35.10.2 Property overview . . . 1163
35.10.3 EHTTP.StatusCode . . . 1163
35.10.4 EHTTP.StatusText . . . 1163
35.11TCookie . . . 1163

35.11.1 Description . . . 1163
35.11.2 Method overview . . . 1163
35.11.3 Property overview . . . 1164
35.11.4 TCookie.Create . . . 1164
35.11.5 TCookie.Assign . . . 1164
35.11.6 TCookie.Expire . . . 1164
35.11.7 TCookie.Name . . . 1164
35.11.8 TCookie.Value . . . 1165
35.11.9 TCookie.Domain . . . 1165
35.11.10 TCookie.Path . . . 1165
35.11.11 TCookie.Expires . . . 1165
35.11.12 TCookie.Secure . . . 1166
35.11.13 TCookie.HttpOnly . . . 1166
35.11.14 TCookie.SameSite . . . 1166
35.11.15 TCookie.AsString . . . 1166
35.12 TCookies . . . 1167
35.12.1 Description . . . 1167
35.12.2 Method overview . . . 1167
35.12.3 Property overview . . . 1167
35.12.4 TCookies.Add . . . 1167
35.12.5 TCookies.CookieByName . . . 1167
35.12.6 TCookies.FindCookie . . . 1167
35.12.7 TCookies.IndexOfCookie . . . 1168
35.12.8 TCookies.Items . . . 1168
35.13 TCORSSupport . . . 1168
35.13.1 Description . . . 1168
35.13.2 Method overview . . . 1168
35.13.3 Property overview . . . 1168
35.13.4 TCORSSupport.Create . . . 1169
35.13.5 TCORSSupport.ResolvedCORSAllowedOrigins . . . 1169
35.13.6 TCORSSupport.HandleRequest . . . 1169
35.13.7 TCORSSupport.Assign . . . 1169
35.13.8 TCORSSupport.Enabled . . . 1169
35.13.9 TCORSSupport.Options . . . 1170
35.13.10 TCORSSupport.AllowedMethods . . . 1170
35.13.11 TCORSSupport.AllowedOrigins . . . 1170
35.13.12 TCORSSupport.AllowedHeaders . . . 1170
35.13.13 TCORSSupport.MaxAge . . . 1171
35.14 TCustomSession . . . 1171
35.14.1 Description . . . 1171

35.14.2 Method overview . . . 1171
35.14.3 Property overview . . . 1171
35.14.4 TCustomSession.Create . . . 1171
35.14.5 TCustomSession.InitSession . . . 1172
35.14.6 TCustomSession.InitResponse . . . 1172
35.14.7 TCustomSession.UpdateResponse . . . 1172
35.14.8 TCustomSession.RemoveVariable . . . 1172
35.14.9 TCustomSession.Terminate . . . 1172
35.14.10TCustomSession.SessionVariableExists . . . 1173
35.14.11TCustomSession.TimeOutMinutes . . . 1173
35.14.12TCustomSession.SessionID . . . 1173
35.14.13TCustomSession.SessionCookie . . . 1173
35.14.14TCustomSession.SessionCookiePath . . . 1174
35.14.15TCustomSession.Variables . . . 1174
35.14.16TCustomSession.SessionState . . . 1174
35.14.17TCustomSession.OnSessionStateChange . . . 1174
35.15THTTPHeader . . . 1174
35.15.1 Description . . . 1174
35.15.2 Method overview . . . 1175
35.15.3 Property overview . . . 1177
35.15.4 THTTPHeader.Create . . . 1178
35.15.5 THTTPHeader.Destroy . . . 1178
35.15.6 THTTPHeader.HeaderIsSet . . . 1178
35.15.7 THTTPHeader.GetHeader . . . 1178
35.15.8 THTTPHeader.SetHeader . . . 1178
35.15.9 THTTPHeader.SetFieldByName . . . 1179
35.15.10THTTPHeader.GetFieldByName . . . 1179
35.15.11THTTPHeader.GetVariableHeaderName . . . 1179
35.15.12THTTPHeader.GetCustomHeader . . . 1179
35.15.13THTTPHeader.SetCustomHeader . . . 1179
35.15.14THTTPHeader.LoadFromStream . . . 1180
35.15.15THTTPHeader.LoadFromStrings . . . 1180
35.15.16THTTPHeader.FieldCount . . . 1180
35.15.17THTTPHeader.Fields . . . 1180
35.15.18THTTPHeader.FieldNames . . . 1180
35.15.19THTTPHeader.FieldValues . . . 1181
35.15.20THTTPHeader.Accept . . . 1181
35.15.21THTTPHeader.AcceptCharset . . . 1181
35.15.22THTTPHeader.AcceptEncoding . . . 1181
35.15.23THTTPHeader.AcceptLanguage . . . 1182

35.15.24 THTTPHeader.Authorization 1182
35.15.25 THTTPHeader.Connection 1182
35.15.26 THTTPHeader.ContentEncoding 1182
35.15.27 THTTPHeader.ContentLanguage 1183
35.15.28 THTTPHeader.ContentLength 1183
35.15.29 THTTPHeader.ContentType 1183
35.15.30 THTTPHeader.Date 1183
35.15.31 THTTPHeader.Expires 1184
35.15.32 THTTPHeader.From 1184
35.15.33 THTTPHeader.Host 1184
35.15.34 THTTPHeader.IfModifiedSince 1184
35.15.35 THTTPHeader.LastModified 1185
35.15.36 THTTPHeader.Location 1185
35.15.37 THTTPHeader.Pragma 1185
35.15.38 THTTPHeader.Referer 1185
35.15.39 THTTPHeader.RetryAfter 1186
35.15.40 THTTPHeader.Server 1186
35.15.41 THTTPHeader.UserAgent 1186
35.15.42 THTTPHeader.Warning 1186
35.15.43 THTTPHeader.WWWAuthenticate 1187
35.15.44 THTTPHeader.Via 1187
35.15.45 THTTPHeader.HTTPAccept 1187
35.15.46 THTTPHeader.HTTPAcceptCharset 1187
35.15.47 THTTPHeader.HTTPAcceptEncoding 1188
35.15.48 THTTPHeader.HTTPIfModifiedSince 1188
35.15.49 THTTPHeader.HTTPReferer 1188
35.15.50 THTTPHeader.HTTPUserAgent 1188
35.15.51 THTTPHeader.Cookie 1189
35.15.52 THTTPHeader.SetCookie 1189
35.15.53 THTTPHeader.HTTPXRequestedWith 1189
35.15.54 THTTPHeader.HttpVersion 1189
35.15.55 THTTPHeader.ProtocolVersion 1190
35.15.56 THTTPHeader.PathInfo 1190
35.15.57 THTTPHeader.PathTranslated 1190
35.15.58 THTTPHeader.RemoteAddress 1190
35.15.59 THTTPHeader.RemoteAddr 1191
35.15.60 THTTPHeader.RemoteHost 1191
35.15.61 THTTPHeader.ScriptName 1191
35.15.62 THTTPHeader.ServerPort 1191
35.15.63 THTTPHeader.Method 1192

35.15.64THTTPHeader.URL 1192
35.15.65THTTPHeader.Query 1192
35.15.66THTTPHeader.Content 1192
35.15.67THTTPHeader.CookieFields 1193
35.15.68THTTPHeader.ContentFields 1193
35.15.69THTTPHeader.QueryFields 1193
35.15.70THTTPHeader.CustomHeaders 1193
35.16TMimeItem 1193
35.16.1 Description 1193
35.16.2 Method overview 1194
35.16.3 Property overview 1194
35.16.4 TMimeItem.Process 1194
35.16.5 TMimeItem.Data 1194
35.16.6 TMimeItem.Name 1194
35.16.7 TMimeItem.Disposition 1195
35.16.8 TMimeItem.FileName 1195
35.16.9 TMimeItem.ContentType 1195
35.16.10TMimeItem.Description 1195
35.16.11TMimeItem.IsFile 1196
35.16.12TMimeItem.DataSize 1196
35.17TMimeItems 1196
35.17.1 Description 1196
35.17.2 Method overview 1196
35.17.3 Property overview 1196
35.17.4 TMimeItems.First 1196
35.17.5 TMimeItems.Last 1197
35.17.6 TMimeItems.Parts 1197
35.18TRequest 1197
35.18.1 Description 1197
35.18.2 Method overview 1197
35.18.3 Property overview 1198
35.18.4 TRequest.Create 1198
35.18.5 TRequest.destroy 1198
35.18.6 TRequest.GetNextPathInfo 1198
35.18.7 TRequest.RouteParams 1199
35.18.8 TRequest.ReturnedPathInfo 1199
35.18.9 TRequest.LocalPathPrefix 1199
35.18.10TRequest.CommandLine 1199
35.18.11TRequest.Command 1200
35.18.12TRequest.URI 1200

35.18.13TRequest.QueryString 1200
35.18.14TRequest.HeaderLine 1200
35.18.15TRequest.Files 1201
35.18.16TRequest.HandleGetOnPost 1201
35.18.17TRequest.OnUnknownEncoding 1201
35.18.18TRequest.IfMatch 1201
35.18.19TRequest.IfNoneMatch 1202
35.18.20TRequest.IfRange 1202
35.18.21TRequest.IfUnModifiedSince 1202
35.18.22TRequest.ContentRange 1202
35.18.23TRequest.TE 1203
35.18.24TRequest.Upgrade 1203
35.19TResponse 1203
35.19.1 Description 1203
35.19.2 Method overview 1203
35.19.3 Property overview 1204
35.19.4 TResponse.Create 1204
35.19.5 TResponse.destroy 1204
35.19.6 TResponse.SendContent 1204
35.19.7 TResponse.SendHeaders 1205
35.19.8 TResponse.SendResponse 1205
35.19.9 TResponse.SendRedirect 1205
35.19.10TResponse.Request 1205
35.19.11TResponse.Code 1205
35.19.12TResponse.CodeText 1206
35.19.13TResponse.Age 1206
35.19.14TResponse.Allow 1206
35.19.15TResponse.CacheControl 1206
35.19.16TResponse.ContentLocation 1207
35.19.17TResponse.ContentMD5 1207
35.19.18TResponse.ContentRange 1207
35.19.19TResponse.ETag 1207
35.19.20TResponse.ProxyAuthenticate 1208
35.19.21TResponse.RetryAfter 1208
35.19.22TResponse.FirstHeaderLine 1208
35.19.23TResponse.ContentStream 1208
35.19.24TResponse.Content 1209
35.19.25TResponse.Contents 1209
35.19.26TResponse.HeadersSent 1209
35.19.27TResponse.ContentSent 1209

35.19.28 TResponse.Cookies . . . 1210
35.19.29 TResponse.FreeContentStream . . . 1210
35.20 TUploadedFile . . . 1210
35.20.1 Description . . . 1210
35.20.2 Method overview . . . 1210
35.20.3 Property overview . . . 1210
35.20.4 TUploadedFile.Destroy . . . 1211
35.20.5 TUploadedFile.FieldName . . . 1211
35.20.6 TUploadedFile.FileName . . . 1211
35.20.7 TUploadedFile.Stream . . . 1211
35.20.8 TUploadedFile.Size . . . 1211
35.20.9 TUploadedFile.ContentType . . . 1212
35.20.10 TUploadedFile.Disposition . . . 1212
35.20.11 TUploadedFile.LocalFileName . . . 1212
35.20.12 TUploadedFile.Description . . . 1212
35.21 TUploadedFiles . . . 1212
35.21.1 Description . . . 1212
35.21.2 Method overview . . . 1213
35.21.3 Property overview . . . 1213
35.21.4 TUploadedFiles.First . . . 1213
35.21.5 TUploadedFiles.Last . . . 1213
35.21.6 TUploadedFiles.IndexOfFile . . . 1213
35.21.7 TUploadedFiles.FileByName . . . 1213
35.21.8 TUploadedFiles.FindFile . . . 1214
35.21.9 TUploadedFiles.Files . . . 1214

36 Reference for unit 'httpprotocol' 1215
36.1 Used units . . . 1215
36.2 Overview . . . 1215
36.3 Constants, types and variables . . . 1215
36.3.1 Constants . . . 1215
36.3.2 Types . . . 1221
36.4 Procedures and functions . . . 1223
36.4.1 ExcludeHTTPPathDelimiter . . . 1223
36.4.2 HeaderName . . . 1223
36.4.3 HeaderType . . . 1224
36.4.4 HTTPDecode . . . 1224
36.4.5 HTTPEncode . . . 1224
36.4.6 IncludeHTTPPathDelimiter . . . 1224

37 Reference for unit 'httproute' 1225

37.1 Used units . . . 1225
37.2 Overview . . . 1225
37.3 Advanced HTTP Router Handler Examples . . . 1225
37.4 HTTP Router Methods and RESTful APIs . . . 1227
37.5 HTTP Router Interceptors and Middleware . . . 1228
37.6 HTTP Router Parameterized Routes . . . 1229
37.7 Basic HTTP Router Usage . . . 1230
37.8 HTTP Router Handler Types . . . 1231
37.9 HTTP Router Working Principle . . . 1231
37.10 Constants, types and variables . . . 1232
37.10.1 Constants . . . 1232
37.10.2 Types . . . 1232
37.11 Procedures and functions . . . 1234
37.11.1 HTTPRouter . . . 1234
37.11.2 RouteMethodToString . . . 1234
37.12 EHTTPRoute . . . 1234
37.12.1 Description . . . 1234
37.13 IRouteInterface . . . 1234
37.13.1 Description . . . 1234
37.13.2 Method overview . . . 1234
37.13.3 IRouteInterface.HandleRequest . . . 1234
37.14 THTTPRoute . . . 1235
37.14.1 Description . . . 1235
37.14.2 Method overview . . . 1235
37.14.3 Property overview . . . 1235
37.14.4 THTTPRoute.Destroy . . . 1235
37.14.5 THTTPRoute.HandleRequest . . . 1235
37.14.6 THTTPRoute.Matches . . . 1235
37.14.7 THTTPRoute.MatchPattern . . . 1236
37.14.8 THTTPRoute.MatchMethod . . . 1236
37.14.9 THTTPRoute.Default . . . 1236
37.14.10 THTTPRoute.URLPattern . . . 1236
37.14.11 THTTPRoute.Method . . . 1237
37.15 THTTPRouteCallback . . . 1237
37.15.1 Description . . . 1237
37.15.2 Property overview . . . 1237
37.15.3 THTTPRouteCallback.CallBack . . . 1237
37.16 THTTPRouteCallbackEx . . . 1237
37.16.1 Description . . . 1237
37.16.2 Property overview . . . 1237

37.16.3 THTTPRouteCallbackEx.CallBack . . . 1238
37.16.4 THTTPRouteCallbackEx.Data . . . 1238
37.17 THTTPRouteEvent . . . 1238
37.17.1 Description . . . 1238
37.17.2 Property overview . . . 1238
37.17.3 THTTPRouteEvent.Event . . . 1238
37.18 THTTPRouteInterface . . . 1238
37.18.1 Description . . . 1238
37.18.2 Property overview . . . 1239
37.18.3 THTTPRouteInterface.Intf . . . 1239
37.19 THTTPRouteList . . . 1239
37.19.1 Description . . . 1239
37.19.2 Property overview . . . 1239
37.19.3 THTTPRouteList.Routes . . . 1239
37.20 THTTPRouteObject . . . 1239
37.20.1 Description . . . 1239
37.20.2 Property overview . . . 1240
37.20.3 THTTPRouteObject.ObjectCLass . . . 1240
37.21 THTTPRouter . . . 1240
37.21.1 Description . . . 1240
37.21.2 Method overview . . . 1240
37.21.3 Property overview . . . 1240
37.21.4 THTTPRouter.Create . . . 1241
37.21.5 THTTPRouter.Destroy . . . 1241
37.21.6 THTTPRouter.DeleteRoute . . . 1241
37.21.7 THTTPRouter.DeleteRouteByID . . . 1241
37.21.8 THTTPRouter.SanitizeRoute . . . 1241
37.21.9 THTTPRouter.Service . . . 1242
37.21.10 THTTPRouter.ServiceClass . . . 1242
37.21.11 THTTPRouter.SetServiceClass . . . 1242
37.21.12 THTTPRouter.StringToRouteMethod . . . 1242
37.21.13 THTTPRouter.RegisterRoute . . . 1242
37.21.14 THTTPRouter.FindHTTPRoute . . . 1243
37.21.15 THTTPRouter.GetHTTPRoute . . . 1243
37.21.16 THTTPRouter.RouteRequest . . . 1244
37.21.17 THTTPRouter.Routes . . . 1244
37.21.18 THTTPRouter.RouteCount . . . 1244
37.21.19 THTTPRouter.BeforeRequest . . . 1244
37.21.20 THTTPRouter.AfterRequest . . . 1245
37.21.21 THTTPRouter.RouteOptions . . . 1245

37.22TRouteObject 1245
37.22.1 Description 1245
37.22.2 Interfaces overview 1245
37.22.3 Method overview 1245
37.22.4 TRouteObject.Create 1245
37.22.5 TRouteObject.HandleRequest 1246

38 Reference for unit 'IBConnection' 1247
38.1 Used units 1247
38.2 Constants, types and variables 1247
38.2.1 Constants 1247
38.2.2 Types 1247
38.3 TDatabaseInfo 1248
38.4 EIBDatabaseError 1248
38.4.1 Description 1248
38.4.2 Property overview 1248
38.4.3 EIBDatabaseError.StatusVector 1248
38.4.4 EIBDatabaseError.GDSErrorCode 1248
38.5 TIBConnection 1249
38.5.1 Description 1249
38.5.2 Method overview 1250
38.5.3 Property overview 1250
38.5.4 TIBConnection.Create 1250
38.5.5 TIBConnection.GetConnectionInfo 1250
38.5.6 TIBConnection.CreateDB 1251
38.5.7 TIBConnection.DropDB 1251
38.5.8 TIBConnection.BlobSegmentSize 1251
38.5.9 TIBConnection.ODSMajorVersion 1252
38.5.10 TIBConnection.DatabaseName 1252
38.5.11 TIBConnection.Dialect 1252
38.5.12 TIBConnection.CheckTransactionParams 1253
38.5.13 TIBConnection.KeepConnection 1253
38.5.14 TIBConnection.LoginPrompt 1253
38.5.15 TIBConnection.Params 1253
38.5.16 TIBConnection.OnLogin 1254
38.5.17 TIBConnection.Port 1254
38.5.18 TIBConnection.UseConnectionCharSetIfNone 1254
38.5.19 TIBConnection.WireCompression 1254
38.6 TIBConnectionDef 1255
38.6.1 Description 1255

38.6.2 Method overview . . . 1255
38.6.3 TIBConnectionDef.TypeName . . . 1255
38.6.4 TIBConnectionDef.ConnectionClass . . . 1255
38.6.5 TIBConnectionDef.Description . . . 1255
38.6.6 TIBConnectionDef.DefaultLibraryName . . . 1256
38.6.7 TIBConnectionDef.LoadFunction . . . 1256
38.6.8 TIBConnectionDef.UnLoadFunction . . . 1256
38.6.9 TIBConnectionDef.LoadedLibraryName . . . 1256
38.7 TIBCursor . . . 1256
38.7.1 Description . . . 1256
38.8 TIBTrans . . . 1257
38.8.1 Description . . . 1257

39 Reference for unit 'idea' 1258
39.1 Used units . . . 1258
39.2 Overview . . . 1258
39.3 Constants, types and variables . . . 1258
39.3.1 Constants . . . 1258
39.3.2 Types . . . 1259
39.4 Procedures and functions . . . 1259
39.4.1 CipherIdea . . . 1259
39.4.2 DeKeyIdea . . . 1259
39.4.3 EnKeyIdea . . . 1260
39.5 EIDEAError . . . 1260
39.5.1 Description . . . 1260
39.6 TIDEADeCryptStream . . . 1260
39.6.1 Description . . . 1260
39.6.2 Method overview . . . 1260
39.6.3 TIDEADeCryptStream.Create . . . 1261
39.6.4 TIDEADeCryptStream.Read . . . 1261
39.6.5 TIDEADeCryptStream.Seek . . . 1261
39.7 TIDEAEncryptStream . . . 1262
39.7.1 Description . . . 1262
39.7.2 Method overview . . . 1262
39.7.3 TIDEAEncryptStream.Create . . . 1262
39.7.4 TIDEAEncryptStream.Destroy . . . 1262
39.7.5 TIDEAEncryptStream.Write . . . 1263
39.7.6 TIDEAEncryptStream.Seek . . . 1263
39.7.7 TIDEAEncryptStream.Flush . . . 1263
39.8 TIDEAStream . . . 1263

39.8.1 Description . . . 1263
39.8.2 Method overview . . . 1264
39.8.3 Property overview . . . 1264
39.8.4 TIDEAStream.Create . . . 1264
39.8.5 TIDEAStream.Key . . . 1264

40 Reference for unit 'inicol' 1265
40.1 Used units . . . 1265
40.2 Overview . . . 1265
40.3 Constants, types and variables . . . 1265
40.3.1 Constants . . . 1265
40.4 EIniCol . . . 1266
40.4.1 Description . . . 1266
40.5 TIniCollection . . . 1266
40.5.1 Description . . . 1266
40.5.2 Method overview . . . 1266
40.5.3 Property overview . . . 1266
40.5.4 TIniCollection.Load . . . 1266
40.5.5 TIniCollection.Save . . . 1267
40.5.6 TIniCollection.SaveToIni . . . 1267
40.5.7 TIniCollection.SaveToFile . . . 1267
40.5.8 TIniCollection.LoadFromIni . . . 1268
40.5.9 TIniCollection.LoadFromFile . . . 1268
40.5.10 TIniCollection.Prefix . . . 1268
40.5.11 TIniCollection.SectionPrefix . . . 1269
40.5.12 TIniCollection.FileName . . . 1269
40.5.13 TIniCollection.GlobalSection . . . 1269
40.6 TIniCollectionItem . . . 1270
40.6.1 Description . . . 1270
40.6.2 Method overview . . . 1270
40.6.3 Property overview . . . 1270
40.6.4 TIniCollectionItem.SaveToIni . . . 1270
40.6.5 TIniCollectionItem.LoadFromIni . . . 1270
40.6.6 TIniCollectionItem.SaveToFile . . . 1271
40.6.7 TIniCollectionItem.LoadFromFile . . . 1271
40.6.8 TIniCollectionItem.SectionName . . . 1271
40.7 TNamedIniCollection . . . 1272
40.7.1 Description . . . 1272
40.7.2 Method overview . . . 1272
40.7.3 Property overview . . . 1272

40.7.4 TNamedIniCollection.IndexOfUserData . . . 1272
40.7.5 TNamedIniCollection.IndexOfName . . . 1272
40.7.6 TNamedIniCollection.FindByName . . . 1273
40.7.7 TNamedIniCollection.FindByUserData . . . 1273
40.7.8 TNamedIniCollection.NamedItems . . . 1273
40.8 TNamedIniCollectionItem . . . 1273
40.8.1 Description . . . 1273
40.8.2 Property overview . . . 1273
40.8.3 TNamedIniCollectionItem.UserData . . . 1274
40.8.4 TNamedIniCollectionItem.Name . . . 1274

41 Reference for unit 'IniFiles' 1275
41.1 Used units . . . 1275
41.2 Overview . . . 1275
41.3 Constants, types and variables . . . 1275
41.3.1 Constants . . . 1275
41.3.2 Types . . . 1275
41.4 TCustomIniFile . . . 1277
41.4.1 Description . . . 1277
41.4.2 Method overview . . . 1277
41.4.3 Property overview . . . 1278
41.4.4 TCustomIniFile.Create . . . 1278
41.4.5 TCustomIniFile.Destroy . . . 1279
41.4.6 TCustomIniFile.SetBoolStringValues . . . 1279
41.4.7 TCustomIniFile.SectionExists . . . 1279
41.4.8 TCustomIniFile.ReadString . . . 1279
41.4.9 TCustomIniFile.WriteString . . . 1280
41.4.10 TCustomIniFile.ReadInteger . . . 1280
41.4.11 TCustomIniFile.WriteInteger . . . 1280
41.4.12 TCustomIniFile.ReadInt64 . . . 1280
41.4.13 TCustomIniFile.WriteInt64 . . . 1281
41.4.14 TCustomIniFile.ReadBool . . . 1281
41.4.15 TCustomIniFile.WriteBool . . . 1281
41.4.16 TCustomIniFile.ReadDate . . . 1282
41.4.17 TCustomIniFile.ReadDateTime . . . 1282
41.4.18 TCustomIniFile.ReadFloat . . . 1282
41.4.19 TCustomIniFile.ReadTime . . . 1282
41.4.20 TCustomIniFile.ReadBinaryStream . . . 1283
41.4.21 TCustomIniFile.WriteDate . . . 1283
41.4.22 TCustomIniFile.WriteDateTime . . . 1283

41.4.23 TCustomIniFile.WriteFloat 1284
41.4.24 TCustomIniFile.WriteTime 1284
41.4.25 TCustomIniFile.WriteBinaryStream 1284
41.4.26 TCustomIniFile.ReadSection 1284
41.4.27 TCustomIniFile.ReadSections 1285
41.4.28 TCustomIniFile.ReadSectionValues 1285
41.4.29 TCustomIniFile.EraseSection 1285
41.4.30 TCustomIniFile.DeleteKey 1286
41.4.31 TCustomIniFile.UpdateFile 1286
41.4.32 TCustomIniFile.ValueExists 1286
41.4.33 TCustomIniFile.Encoding 1286
41.4.34 TCustomIniFile.FileName 1287
41.4.35 TCustomIniFile.Options 1287
41.4.36 TCustomIniFile.EscapeLineFeeds 1287
41.4.37 TCustomIniFile.CaseSensitive 1287
41.4.38 TCustomIniFile.StripQuotes 1288
41.4.39 TCustomIniFile.FormatSettingsActive 1288
41.4.40 TCustomIniFile.BoolTrueStrings 1288
41.4.41 TCustomIniFile.BoolFalseStrings 1289
41.4.42 TCustomIniFile.OwnsEncoding 1289
41.5 THashedStringList 1289
41.5.1 Description 1289
41.5.2 Method overview 1289
41.5.3 THashedStringList.Destroy 1289
41.5.4 THashedStringList.IndexOf 1290
41.5.5 THashedStringList.IndexOfName 1290
41.6 TIniFile 1290
41.6.1 Description 1290
41.6.2 Method overview 1290
41.6.3 Property overview 1291
41.6.4 TIniFile.Create 1291
41.6.5 TIniFile.Destroy 1291
41.6.6 TIniFile.ReadString 1291
41.6.7 TIniFile.WriteString 1292
41.6.8 TIniFile.ReadSection 1292
41.6.9 TIniFile.ReadSectionRaw 1292
41.6.10 TIniFile.ReadSections 1292
41.6.11 TIniFile.ReadSectionValues 1293
41.6.12 TIniFile.EraseSection 1293
41.6.13 TIniFile.DeleteKey 1293

41.6.14 TIniFile.UpdateFile . . . 1293
41.6.15 TIniFile.Stream . . . 1294
41.6.16 TIniFile.CacheUpdates . . . 1294
41.6.17 TIniFile.WriteBOM . . . 1294
41.7 TIniFileKey . . . 1295
41.7.1 Description . . . 1295
41.7.2 Method overview . . . 1295
41.7.3 Property overview . . . 1295
41.7.4 TIniFileKey.Create . . . 1295
41.7.5 TIniFileKey.Ident . . . 1295
41.7.6 TIniFileKey.Value . . . 1295
41.8 TIniFileKeyList . . . 1296
41.8.1 Description . . . 1296
41.8.2 Method overview . . . 1296
41.8.3 Property overview . . . 1296
41.8.4 TIniFileKeyList.Destroy . . . 1296
41.8.5 TIniFileKeyList.Clear . . . 1296
41.8.6 TIniFileKeyList.Items . . . 1296
41.9 TIniFileSection . . . 1297
41.9.1 Description . . . 1297
41.9.2 Method overview . . . 1297
41.9.3 Property overview . . . 1297
41.9.4 TIniFileSection.Empty . . . 1297
41.9.5 TIniFileSection.Create . . . 1297
41.9.6 TIniFileSection.Destroy . . . 1298
41.9.7 TIniFileSection.Name . . . 1298
41.9.8 TIniFileSection.KeyList . . . 1298
41.10 TIniFileSectionList . . . 1298
41.10.1 Description . . . 1298
41.10.2 Method overview . . . 1298
41.10.3 Property overview . . . 1299
41.10.4 TIniFileSectionList.Destroy . . . 1299
41.10.5 TIniFileSectionList.Clear . . . 1299
41.10.6 TIniFileSectionList.Items . . . 1299
41.11 TMemIniFile . . . 1299
41.11.1 Description . . . 1299
41.11.2 Method overview . . . 1300
41.11.3 TMemIniFile.Create . . . 1300
41.11.4 TMemIniFile.Clear . . . 1300
41.11.5 TMemIniFile.GetStrings . . . 1300

41.11.6 TMemIniFile.Rename . . . 1301
41.11.7 TMemIniFile.SetStrings . . . 1301
41.12TStringHash . . . 1301
41.12.1 Description . . . 1301
41.12.2 Method overview . . . 1301
41.12.3 Property overview . . . 1301
41.12.4 TStringHash.Create . . . 1302
41.12.5 TStringHash.Destroy . . . 1302
41.12.6 TStringHash.Add . . . 1302
41.12.7 TStringHash.Clear . . . 1302
41.12.8 TStringHash.Modify . . . 1303
41.12.9 TStringHash.Remove . . . 1303
41.12.10TStringHash.ValueOf . . . 1303
41.12.11TStringHash.AddReplacesExisting . . . 1303

42 Reference for unit 'iostream' 1304
42.1 Used units . . . 1304
42.2 Overview . . . 1304
42.3 Constants, types and variables . . . 1304
42.3.1 Types . . . 1304
42.4 EIOStreamError . . . 1305
42.4.1 Description . . . 1305
42.5 TIOStream . . . 1305
42.5.1 Description . . . 1305
42.5.2 Method overview . . . 1305
42.5.3 TIOStream.Create . . . 1305
42.5.4 TIOStream.Read . . . 1305
42.5.5 TIOStream.Write . . . 1306
42.5.6 TIOStream.Seek . . . 1306

43 Reference for unit 'json2yaml' 1307
43.1 Used units . . . 1307
43.2 Overview . . . 1307
43.3 TJSONToYaml . . . 1307
43.3.1 Description . . . 1307
43.3.2 Method overview . . . 1308
43.3.3 Property overview . . . 1308
43.3.4 TJSONToYaml.Convert . . . 1308
43.3.5 TJSONToYaml.AddHeader . . . 1308
43.3.6 TJSONToYaml.IndentSize . . . 1308

44 Reference for unit 'jsonini' 1310
44.1 Used units . . . 1310
44.2 Overview . . . 1310
44.3 Basic usage of the JSON ini file class . . . 1310
44.4 TJSONIniFile . . . 1312
44.4.1 Description . . . 1312
44.4.2 Method overview . . . 1313
44.4.3 Property overview . . . 1313
44.4.4 TJSONIniFile.Create . . . 1313
44.4.5 TJSONIniFile.Destroy . . . 1314
44.4.6 TJSONIniFile.ConvertIni . . . 1314
44.4.7 TJSONIniFile.ReadString . . . 1314
44.4.8 TJSONIniFile.ReadInteger . . . 1314
44.4.9 TJSONIniFile.ReadInt64 . . . 1315
44.4.10 TJSONIniFile.ReadBool . . . 1315
44.4.11 TJSONIniFile.ReadDate . . . 1315
44.4.12 TJSONIniFile.ReadDateTime . . . 1315
44.4.13 TJSONIniFile.ReadFloat . . . 1316
44.4.14 TJSONIniFile.ReadTime . . . 1316
44.4.15 TJSONIniFile.WriteString . . . 1316
44.4.16 TJSONIniFile.WriteDate . . . 1316
44.4.17 TJSONIniFile.WriteDateTime . . . 1317
44.4.18 TJSONIniFile.WriteFloat . . . 1317
44.4.19 TJSONIniFile.WriteTime . . . 1317
44.4.20 TJSONIniFile.WriteInteger . . . 1317
44.4.21 TJSONIniFile.WriteInt64 . . . 1318
44.4.22 TJSONIniFile.WriteBool . . . 1318
44.4.23 TJSONIniFile.ReadSection . . . 1318
44.4.24 TJSONIniFile.ReadSections . . . 1318
44.4.25 TJSONIniFile.ReadSectionValues . . . 1319
44.4.26 TJSONIniFile.EraseSection . . . 1319
44.4.27 TJSONIniFile.DeleteKey . . . 1319
44.4.28 TJSONIniFile.UpdateFile . . . 1319
44.4.29 TJSONIniFile.Stream . . . 1320
44.4.30 TJSONIniFile.CacheUpdates . . . 1320

45 Reference for unit 'jsonparser' 1321
45.1 Used units . . . 1321
45.2 Overview . . . 1321
45.3 Constants, types and variables . . . 1321

45.3.1 Types . . . 1321
45.4 TJSONParser . . . 1321
45.4.1 Description . . . 1321
45.4.2 Method overview . . . 1322
45.4.3 TJSONParser.Parse . . . 1322

46 Reference for unit 'jsonreader' 1323
46.1 Used units . . . 1323
46.2 Overview . . . 1323
46.3 Using JSON readers with streams and files . . . 1323
46.4 Using TJSONConsumerReader with IJSONConsumer interface . . . 1326
46.5 Using TJSONEventReader for event-based JSON parsing . . . 1329
46.6 Constants, types and variables . . . 1331
46.6.1 Types . . . 1331
46.7 EJSONParser . . . 1332
46.7.1 Description . . . 1332
46.8 IJSONConsumer . . . 1332
46.8.1 Description . . . 1332
46.8.2 Method overview . . . 1332
46.8.3 IJSONConsumer.NullValue . . . 1333
46.8.4 IJSONConsumer.BooleanValue . . . 1333
46.8.5 IJSONConsumer.NumberValue . . . 1333
46.8.6 IJSONConsumer.FloatValue . . . 1333
46.8.7 IJSONConsumer.Int64Value . . . 1334
46.8.8 IJSONConsumer.QWordValue . . . 1334
46.8.9 IJSONConsumer.IntegerValue . . . 1334
46.8.10 IJSONConsumer.StringValue . . . 1334
46.8.11 IJSONConsumer.KeyName . . . 1335
46.8.12 IJSONConsumer.StartObject . . . 1335
46.8.13 IJSONConsumer.EndObject . . . 1335
46.8.14 IJSONConsumer.StartArray . . . 1335
46.8.15 IJSONConsumer.EndArray . . . 1336
46.9 TBaseJSONReader . . . 1336
46.9.1 Description . . . 1336
46.9.2 Method overview . . . 1336
46.9.3 Property overview . . . 1336
46.9.4 TBaseJSONReader.Create . . . 1336
46.9.5 TBaseJSONReader.Destroy . . . 1337
46.9.6 TBaseJSONReader.Options . . . 1337
46.10TJSONConsumerReader . . . 1337

46.10.1 Description . . . 1337
46.10.2 Method overview . . . 1337
46.10.3 Property overview . . . 1338
46.10.4 TJSONConsumerReader.Execute . . . 1338
46.10.5 TJSONConsumerReader.Consumer . . . 1338
46.11 TJSONEventReader . . . 1338
46.11.1 Description . . . 1338
46.11.2 Method overview . . . 1338
46.11.3 Property overview . . . 1339
46.11.4 TJSONEventReader.Execute . . . 1339
46.11.5 TJSONEventReader.OnNullValue . . . 1339
46.11.6 TJSONEventReader.OnBooleanValue . . . 1339
46.11.7 TJSONEventReader.OnNumberValue . . . 1340
46.11.8 TJSONEventReader.OnFloatValue . . . 1340
46.11.9 TJSONEventReader.OnIntegerValue . . . 1340
46.11.10 TJSONEventReader.OnInt64Value . . . 1341
46.11.11 TJSONEventReader.OnQWordValue . . . 1341
46.11.12 TJSONEventReader.OnStringValue . . . 1341
46.11.13 TJSONEventReader.OnKeyName . . . 1341
46.11.14 TJSONEventReader.OnStartObject . . . 1342
46.11.15 TJSONEventReader.OnEndObject . . . 1342
46.11.16 TJSONEventReader.OnStartArray . . . 1342
46.11.17 TJSONEventReader.OnEndArray . . . 1343

47 Reference for unit 'jsonscanner' 1344
47.1 Used units . . . 1344
47.2 Overview . . . 1344
47.3 Constants, types and variables . . . 1344
47.3.1 Resource strings . . . 1344
47.3.2 Constants . . . 1345
47.3.3 Types . . . 1345
47.4 EScannerError . . . 1347
47.4.1 Description . . . 1347
47.5 TJSONScanner . . . 1347
47.5.1 Description . . . 1347
47.5.2 Method overview . . . 1347
47.5.3 Property overview . . . 1347
47.5.4 TJSONScanner.Create . . . 1347
47.5.5 TJSONScanner.FetchToken . . . 1348
47.5.6 TJSONScanner.CurLine . . . 1348

47.5.7 TJSONScanner.CurRow . . . 1348
47.5.8 TJSONScanner.CurColumn . . . 1349
47.5.9 TJSONScanner.CurToken . . . 1349
47.5.10 TJSONScanner.CurTokenString . . . 1349
47.5.11 TJSONScanner.Strict . . . 1349
47.5.12 TJSONScanner.UseUTF8 . . . 1350
47.5.13 TJSONScanner.Options . . . 1350

48 Reference for unit 'libtar' 1351
48.1 Used units . . . 1351
48.2 Overview . . . 1351
48.3 Constants, types and variables . . . 1351
48.3.1 Constants . . . 1351
48.3.2 Types . . . 1352
48.4 Procedures and functions . . . 1354
48.4.1 ClearDirRec . . . 1354
48.4.2 ConvertFilename . . . 1354
48.4.3 FileTimeGMT . . . 1354
48.4.4 PermissionString . . . 1354
48.5 TTarArchive . . . 1355
48.5.1 Description . . . 1355
48.5.2 Method overview . . . 1355
48.5.3 TTarArchive.Create . . . 1355
48.5.4 TTarArchive.Destroy . . . 1355
48.5.5 TTarArchive.Reset . . . 1356
48.5.6 TTarArchive.FindNext . . . 1356
48.5.7 TTarArchive.ReadFile . . . 1356
48.5.8 TTarArchive.GetFilePos . . . 1357
48.5.9 TTarArchive.SetFilePos . . . 1357
48.6 TTarWriter . . . 1357
48.6.1 Description . . . 1357
48.6.2 Method overview . . . 1357
48.6.3 Property overview . . . 1358
48.6.4 TTarWriter.Create . . . 1358
48.6.5 TTarWriter.Destroy . . . 1358
48.6.6 TTarWriter.AddFile . . . 1358
48.6.7 TTarWriter.AddStream . . . 1359
48.6.8 TTarWriter.AddString . . . 1359
48.6.9 TTarWriter.AddDir . . . 1359
48.6.10 TTarWriter.AddSymbolicLink . . . 1360

48.6.11 TTarWriter.AddLink . . . 1360
48.6.12 TTarWriter.AddVolumeHeader . . . 1360
48.6.13 TTarWriter.Finalize . . . 1360
48.6.14 TTarWriter.Permissions . . . 1361
48.6.15 TTarWriter.UID . . . 1361
48.6.16 TTarWriter.GID . . . 1361
48.6.17 TTarWriter.UserName . . . 1361
48.6.18 TTarWriter.GroupName . . . 1362
48.6.19 TTarWriter.Mode . . . 1362
48.6.20 TTarWriter.Magic . . . 1362

49 Reference for unit 'MaskUtils' 1363
49.1 Used units . . . 1363
49.2 Overview . . . 1363
49.3 Constants, types and variables . . . 1363
49.3.1 Types . . . 1363
49.4 Procedures and functions . . . 1365
49.4.1 FormatMaskInput . . . 1365
49.4.2 FormatMaskText . . . 1366
49.4.3 MaskDoFormatText . . . 1366
49.5 TMaskUtils . . . 1366
49.5.1 Description . . . 1366
49.5.2 Method overview . . . 1366
49.5.3 Property overview . . . 1366
49.5.4 TMaskUtils.ValidateInput . . . 1367
49.5.5 TMaskUtils.TryValidateInput . . . 1367
49.5.6 TMaskUtils.Mask . . . 1367
49.5.7 TMaskUtils.Value . . . 1367
49.5.8 TMaskUtils.InputMask . . . 1368

50 Reference for unit 'memds' 1369
50.1 Used units . . . 1369
50.2 Overview . . . 1369
50.3 Constants, types and variables . . . 1369
50.3.1 Constants . . . 1369
50.4 MDSError . . . 1370
50.4.1 Description . . . 1370
50.5 TMemDataset . . . 1371
50.5.1 Description . . . 1371
50.5.2 Method overview . . . 1372
50.5.3 Property overview . . . 1373

50.5.4 TMemDataset.Create 1373
50.5.5 TMemDataset.Destroy 1373
50.5.6 TMemDataset.BookmarkValid 1374
50.5.7 TMemDataset.CompareBookmarks 1374
50.5.8 TMemDataset.CreateBlobStream 1374
50.5.9 TMemDataset.Locate 1375
50.5.10 TMemDataset.Lookup 1375
50.5.11 TMemDataset.CreateTable 1375
50.5.12 TMemDataset.DataSize 1376
50.5.13 TMemDataset.Clear 1376
50.5.14 TMemDataset.SaveToFile 1376
50.5.15 TMemDataset.SaveToStream 1377
50.5.16 TMemDataset.LoadFromStream 1377
50.5.17 TMemDataset.LoadFromFile 1377
50.5.18 TMemDataset.CopyFromDataset 1378
50.5.19 TMemDataset.FileModified 1379
50.5.20 TMemDataset.Filter 1379
50.5.21 TMemDataset.FileName 1379
50.5.22 TMemDataset.Filtered 1380
50.5.23 TMemDataset.Active 1380
50.5.24 TMemDataset.FieldDefs 1380
50.5.25 TMemDataset.BeforeOpen 1380
50.5.26 TMemDataset.AfterOpen 1380
50.5.27 TMemDataset.BeforeClose 1381
50.5.28 TMemDataset.AfterClose 1381
50.5.29 TMemDataset.BeforeInsert 1381
50.5.30 TMemDataset.AfterInsert 1381
50.5.31 TMemDataset.BeforeEdit 1381
50.5.32 TMemDataset.AfterEdit 1381
50.5.33 TMemDataset.BeforePost 1382
50.5.34 TMemDataset.AfterPost 1382
50.5.35 TMemDataset.BeforeCancel 1382
50.5.36 TMemDataset.AfterCancel 1382
50.5.37 TMemDataset.BeforeDelete 1382
50.5.38 TMemDataset.AfterDelete 1382
50.5.39 TMemDataset.BeforeScroll 1383
50.5.40 TMemDataset.AfterScroll 1383
50.5.41 TMemDataset.OnDeleteError 1383
50.5.42 TMemDataset.OnEditError 1383
50.5.43 TMemDataset.OnNewRecord 1383

50.5.44 TMemDataset.OnPostError 1384
50.5.45 TMemDataset.OnFilterRecord 1384

51 Reference for unit 'MSSQLConn' 1385
51.1 Used units 1385
51.2 Overview 1385
51.3 Constants, types and variables 1385
51.3.1 Variables 1385
51.4 EMSSQLDatabaseError 1385
51.4.1 Description 1385
51.4.2 Property overview 1386
51.4.3 EMSSQLDatabaseError.DBErrorCode 1386
51.5 TMSSQLConnection 1386
51.5.1 Description 1386
51.5.2 Method overview 1387
51.5.3 Property overview 1387
51.5.4 TMSSQLConnection.Create 1387
51.5.5 TMSSQLConnection.GetConnectionInfo 1387
51.5.6 TMSSQLConnection.CreateDB 1387
51.5.7 TMSSQLConnection.DropDB 1388
51.5.8 TMSSQLConnection.Password 1388
51.5.9 TMSSQLConnection.Transaction 1388
51.5.10 TMSSQLConnection.UserName 1388
51.5.11 TMSSQLConnection.CharSet 1389
51.5.12 TMSSQLConnection.HostName 1389
51.5.13 TMSSQLConnection.Connected 1389
51.5.14 TMSSQLConnection.Role 1389
51.5.15 TMSSQLConnection.DatabaseName 1390
51.5.16 TMSSQLConnection.KeepConnection 1390
51.5.17 TMSSQLConnection.LoginPrompt 1390
51.5.18 TMSSQLConnection.Params 1390
51.5.19 TMSSQLConnection.OnLogin 1391
51.6 TMSSQLConnectionDef 1391
51.6.1 Description 1391
51.6.2 Method overview 1391
51.6.3 TMSSQLConnectionDef.TypeName 1391
51.6.4 TMSSQLConnectionDef.ConnectionClass 1391
51.6.5 TMSSQLConnectionDef.Description 1392
51.6.6 TMSSQLConnectionDef.DefaultLibraryName 1392
51.6.7 TMSSQLConnectionDef.LoadFunction 1392

51.6.8 TMSSQLConnectionDef.UnLoadFunction . . . 1392
51.6.9 TMSSQLConnectionDef.LoadedLibraryName . . . 1392
51.7 TSybaseConnection . . . 1393
51.7.1 Description . . . 1393
51.7.2 Method overview . . . 1393
51.7.3 TSybaseConnection.Create . . . 1393
51.8 TSybaseConnectionDef . . . 1393
51.8.1 Description . . . 1393
51.8.2 Method overview . . . 1393
51.8.3 TSybaseConnectionDef.TypeName . . . 1393
51.8.4 TSybaseConnectionDef.ConnectionClass . . . 1394
51.8.5 TSybaseConnectionDef.Description . . . 1394

52 Reference for unit 'nullstream' 1395
52.1 Used units . . . 1395
52.2 Overview . . . 1395
52.3 ENullStreamError . . . 1395
52.3.1 Description . . . 1395
52.4 TNullStream . . . 1395
52.4.1 Description . . . 1395
52.4.2 Method overview . . . 1396
52.4.3 TNullStream.Read . . . 1396
52.4.4 TNullStream.Write . . . 1396
52.4.5 TNullStream.Seek . . . 1396
52.4.6 TNullStream.Create . . . 1397

53 Reference for unit 'pascodegen' 1398
53.1 Used units . . . 1398
53.2 Overview . . . 1398
53.3 Constants, types and variables . . . 1398
53.3.1 Types . . . 1398
53.4 TPascalCodeGenerator . . . 1399
53.4.1 Description . . . 1399
53.4.2 Method overview . . . 1400
53.4.3 Property overview . . . 1400
53.4.4 TPascalCodeGenerator.Create . . . 1400
53.4.5 TPascalCodeGenerator.Destroy . . . 1401
53.4.6 TPascalCodeGenerator.EnsureSection . . . 1401
53.4.7 TPascalCodeGenerator.PushSection . . . 1401
53.4.8 TPascalCodeGenerator.PopSection . . . 1401
53.4.9 TPascalCodeGenerator.CreateHeader . . . 1402

53.4.10 TPascalCodeGenerator.CreateUnitClause . . . 1402
53.4.11 TPascalCodeGenerator.Indent . . . 1402
53.4.12 TPascalCodeGenerator.Undent . . . 1402
53.4.13 TPascalCodeGenerator.IsKeyWord . . . 1403
53.4.14 TPascalCodeGenerator.EscapeKeyWord . . . 1403
53.4.15 TPascalCodeGenerator.MakePascalString . . . 1403
53.4.16 TPascalCodeGenerator.PrettyPrint . . . 1403
53.4.17 TPascalCodeGenerator.AddLn . . . 1404
53.4.18 TPascalCodeGenerator.Comment . . . 1404
53.4.19 TPascalCodeGenerator.ClassHeader . . . 1404
53.4.20 TPascalCodeGenerator.SimpleMethodBody . . . 1404
53.4.21 TPascalCodeGenerator.SaveToStream . . . 1405
53.4.22 TPascalCodeGenerator.SaveToFile . . . 1405
53.4.23 TPascalCodeGenerator.Source . . . 1405
53.4.24 TPascalCodeGenerator.CurrentSection . . . 1405
53.4.25 TPascalCodeGenerator.OutputUnitName . . . 1406
53.4.26 TPascalCodeGenerator.ExtraUnits . . . 1406
53.4.27 TPascalCodeGenerator.LicenseText . . . 1406
53.4.28 TPascalCodeGenerator.Switches . . . 1406
53.4.29 TPascalCodeGenerator.OnLog . . . 1407
53.4.30 TPascalCodeGenerator.AddTimeStamp . . . 1407
53.4.31 TPascalCodeGenerator.KeywordSuffix . . . 1407
53.4.32 TPascalCodeGenerator.KeywordPrefix . . . 1407

54 Reference for unit 'Pipes' 1408
54.1 Used units . . . 1408
54.2 Overview . . . 1408
54.3 Constants, types and variables . . . 1408
54.3.1 Constants . . . 1408
54.4 Procedures and functions . . . 1408
54.4.1 CreatePipeHandles . . . 1408
54.4.2 CreatePipeStreams . . . 1409
54.5 EPipeCreation . . . 1409
54.5.1 Description . . . 1409
54.6 EPipeError . . . 1409
54.6.1 Description . . . 1409
54.7 EPipeSeek . . . 1409
54.7.1 Description . . . 1409
54.8 TInputPipeStream . . . 1410
54.8.1 Description . . . 1410

54.8.2 Method overview . . . 1410
54.8.3 Property overview . . . 1410
54.8.4 TInputPipeStream.Destroy . . . 1410
54.8.5 TInputPipeStream.Write . . . 1410
54.8.6 TInputPipeStream.Seek . . . 1411
54.8.7 TInputPipeStream.Read . . . 1411
54.8.8 TInputPipeStream.NumBytesAvailable . . . 1411
54.9 TOutputPipeStream . . . 1412
54.9.1 Description . . . 1412
54.9.2 Method overview . . . 1412
54.9.3 Property overview . . . 1412
54.9.4 TOutputPipeStream.Destroy . . . 1412
54.9.5 TOutputPipeStream.Seek . . . 1412
54.9.6 TOutputPipeStream.Read . . . 1412
54.9.7 TOutputPipeStream.DontClose . . . 1413

55 Reference for unit 'pooledmm' 1414
55.1 Used units . . . 1414
55.2 Overview . . . 1414
55.3 Constants, types and variables . . . 1414
55.3.1 Types . . . 1414
55.4 TPooledMemManagerItem . . . 1414
55.5 TNonFreePooledMemManager . . . 1415
55.5.1 Description . . . 1415
55.5.2 Method overview . . . 1415
55.5.3 Property overview . . . 1415
55.5.4 TNonFreePooledMemManager.Clear . . . 1415
55.5.5 TNonFreePooledMemManager.Create . . . 1415
55.5.6 TNonFreePooledMemManager.Destroy . . . 1416
55.5.7 TNonFreePooledMemManager.NewItem . . . 1416
55.5.8 TNonFreePooledMemManager.EnumerateItems . . . 1416
55.5.9 TNonFreePooledMemManager.ItemSize . . . 1416
55.6 TPooledMemManager . . . 1417
55.6.1 Description . . . 1417
55.6.2 Method overview . . . 1417
55.6.3 Property overview . . . 1417
55.6.4 TPooledMemManager.Clear . . . 1417
55.6.5 TPooledMemManager.Create . . . 1417
55.6.6 TPooledMemManager.Destroy . . . 1418
55.6.7 TPooledMemManager.MinimumFreeCount . . . 1418

55.6.8 TPooledMemManager.MaximumFreeCountRatio . . . 1418
55.6.9 TPooledMemManager.Count . . . 1418
55.6.10 TPooledMemManager.FreeCount . . . 1419
55.6.11 TPooledMemManager.AllocatedCount . . . 1419
55.6.12 TPooledMemManager.FreedCount . . . 1419

56 Reference for unit 'process' 1420
56.1 Used units . . . 1420
56.2 Overview . . . 1420
56.3 Constants, types and variables . . . 1420
56.3.1 Types . . . 1420
56.3.2 Variables . . . 1424
56.4 Procedures and functions . . . 1424
56.4.1 CommandToList . . . 1424
56.4.2 DetectXTerm . . . 1424
56.4.3 RunCommand . . . 1425
56.4.4 RunCommandIndir . . . 1425
56.5 EProcess . . . 1426
56.5.1 Description . . . 1426
56.6 TPROCESS . . . 1426
56.6.1 Description . . . 1426
56.6.2 Method overview . . . 1426
56.6.3 Property overview . . . 1427
56.6.4 TPROCESS.Create . . . 1428
56.6.5 TPROCESS.Destroy . . . 1428
56.6.6 TPROCESS.Execute . . . 1428
56.6.7 TPROCESS.CloseInput . . . 1429
56.6.8 TPROCESS.CloseOutput . . . 1429
56.6.9 TPROCESS.CloseStderr . . . 1429
56.6.10 TPROCESS.Resume . . . 1429
56.6.11 TPROCESS.Suspend . . . 1430
56.6.12 TPROCESS.Terminate . . . 1430
56.6.13 TPROCESS.WaitOnExit . . . 1430
56.6.14 TPROCESS.ReadInputStream . . . 1431
56.6.15 TPROCESS.RunCommandLoop . . . 1431
56.6.16 TPROCESS.WindowRect . . . 1431
56.6.17 TPROCESS.Handle . . . 1432
56.6.18 TPROCESS.ProcessHandle . . . 1432
56.6.19 TPROCESS.ThreadHandle . . . 1432
56.6.20 TPROCESS.ProcessID . . . 1432

56.6.21 TPROCESS.ThreadID 1433
56.6.22 TPROCESS.Input 1433
56.6.23 TPROCESS.Output 1433
56.6.24 TPROCESS.Stderr 1434
56.6.25 TPROCESS.ExitStatus 1434
56.6.26 TPROCESS.ExitCode 1435
56.6.27 TPROCESS.InheritHandles 1435
56.6.28 TPROCESS.OnRunCommandEvent 1435
56.6.29 TPROCESS.RunCommandSleepTime 1435
56.6.30 TPROCESS.OnForkEvent 1436
56.6.31 TPROCESS.PipeBufferSize 1436
56.6.32 TPROCESS.Active 1436
56.6.33 TPROCESS.ApplicationName 1436
56.6.34 TPROCESS.CommandLine 1437
56.6.35 TPROCESS.Executable 1437
56.6.36 TPROCESS.Parameters 1438
56.6.37 TPROCESS.ConsoleTitle 1438
56.6.38 TPROCESS.CurrentDirectory 1439
56.6.39 TPROCESS.Desktop 1439
56.6.40 TPROCESS.Environment 1439
56.6.41 TPROCESS.Options 1440
56.6.42 TPROCESS.Priority 1440
56.6.43 TPROCESS.StartupOptions 1441
56.6.44 TPROCESS.Running 1441
56.6.45 TPROCESS.ShowWindow 1442
56.6.46 TPROCESS.WindowColumns 1442
56.6.47 TPROCESS.WindowHeight 1442
56.6.48 TPROCESS.WindowLeft 1443
56.6.49 TPROCESS.WindowRows 1443
56.6.50 TPROCESS.WindowTop 1443
56.6.51 TPROCESS.WindowWidth 1444
56.6.52 TPROCESS.FillAttribute 1444
56.6.53 TPROCESS.XTermProgram 1444

57 Reference for unit 'RttiUtils' 1445

57.1 Used units 1445
57.2 Overview 1445
57.3 Constants, types and variables 1445
57.3.1 Constants 1445
57.3.2 Types 1445

57.3.3 Variables . . . 1446
57.4 Procedures and functions . . . 1447
57.4.1 CreateStoredItem . . . 1447
57.4.2 ParseStoredItem . . . 1447
57.4.3 UpdateStoredList . . . 1447
57.5 TPropInfoList . . . 1448
57.5.1 Description . . . 1448
57.5.2 Method overview . . . 1448
57.5.3 Property overview . . . 1448
57.5.4 TPropInfoList.Create . . . 1448
57.5.5 TPropInfoList.Destroy . . . 1448
57.5.6 TPropInfoList.Contains . . . 1449
57.5.7 TPropInfoList.Find . . . 1449
57.5.8 TPropInfoList.Delete . . . 1449
57.5.9 TPropInfoList.Intersect . . . 1449
57.5.10 TPropInfoList.Count . . . 1450
57.5.11 TPropInfoList.Items . . . 1450
57.6 TPropsStorage . . . 1450
57.6.1 Description . . . 1450
57.6.2 Method overview . . . 1450
57.6.3 Property overview . . . 1451
57.6.4 TPropsStorage.StoreAnyProperty . . . 1451
57.6.5 TPropsStorage.LoadAnyProperty . . . 1451
57.6.6 TPropsStorage.StoreProperties . . . 1451
57.6.7 TPropsStorage.LoadProperties . . . 1452
57.6.8 TPropsStorage.LoadObjectsProps . . . 1452
57.6.9 TPropsStorage.StoreObjectsProps . . . 1452
57.6.10 TPropsStorage.Options . . . 1453
57.6.11 TPropsStorage.AObject . . . 1453
57.6.12 TPropsStorage.Prefix . . . 1454
57.6.13 TPropsStorage.Section . . . 1454
57.6.14 TPropsStorage.OnReadString . . . 1454
57.6.15 TPropsStorage.OnWriteString . . . 1454
57.6.16 TPropsStorage.OnEraseSection . . . 1455

58 Reference for unit 'simpleipc' 1456
58.1 Used units . . . 1456
58.2 Overview . . . 1456
58.3 Constants, types and variables . . . 1457
58.3.1 Resource strings . . . 1457

58.3.2 Constants . . . 1457
58.3.3 Types . . . 1457
58.3.4 Variables . . . 1458
58.4 TMsgHeader . . . 1459
58.5 EIPCError . . . 1459
58.5.1 Description . . . 1459
58.6 TIPCClientComm . . . 1459
58.6.1 Description . . . 1459
58.6.2 Method overview . . . 1460
58.6.3 Property overview . . . 1460
58.6.4 TIPCClientComm.Create . . . 1460
58.6.5 TIPCClientComm.Connect . . . 1460
58.6.6 TIPCClientComm.Disconnect . . . 1460
58.6.7 TIPCClientComm.ServerRunning . . . 1461
58.6.8 TIPCClientComm.SendMessage . . . 1461
58.6.9 TIPCClientComm.Owner . . . 1461
58.7 TIPCServerComm . . . 1462
58.7.1 Description . . . 1462
58.7.2 Method overview . . . 1462
58.7.3 Property overview . . . 1462
58.7.4 TIPCServerComm.Create . . . 1462
58.7.5 TIPCServerComm.StartServer . . . 1462
58.7.6 TIPCServerComm.StopServer . . . 1463
58.7.7 TIPCServerComm.PeekMessage . . . 1463
58.7.8 TIPCServerComm.ReadMessage . . . 1463
58.7.9 TIPCServerComm.Owner . . . 1464
58.7.10 TIPCServerComm.InstanceID . . . 1464
58.8 TIPCServerMsg . . . 1464
58.8.1 Description . . . 1464
58.8.2 Method overview . . . 1464
58.8.3 Property overview . . . 1464
58.8.4 TIPCServerMsg.Create . . . 1465
58.8.5 TIPCServerMsg.Destroy . . . 1465
58.8.6 TIPCServerMsg.Stream . . . 1465
58.8.7 TIPCServerMsg.MsgType . . . 1465
58.8.8 TIPCServerMsg.OwnsStream . . . 1466
58.8.9 TIPCServerMsg.StringMessage . . . 1466
58.9 TIPCServerMsgQueue . . . 1466
58.9.1 Description . . . 1466
58.9.2 Method overview . . . 1466

58.9.3 Property overview . . . 1466
58.9.4 TIPCServerMsgQueue.Create . . . 1467
58.9.5 TIPCServerMsgQueue.Destroy . . . 1467
58.9.6 TIPCServerMsgQueue.Clear . . . 1467
58.9.7 TIPCServerMsgQueue.Push . . . 1467
58.9.8 TIPCServerMsgQueue.Pop . . . 1468
58.9.9 TIPCServerMsgQueue.Count . . . 1468
58.9.10 TIPCServerMsgQueue.MaxCount . . . 1468
58.9.11 TIPCServerMsgQueue.MaxAction . . . 1468
58.10 TSimpleIPC . . . 1469
58.10.1 Description . . . 1469
58.10.2 Property overview . . . 1469
58.10.3 TSimpleIPC.Active . . . 1469
58.10.4 TSimpleIPC.ServerID . . . 1469
58.11 TSimpleIPCClient . . . 1470
58.11.1 Description . . . 1470
58.11.2 Method overview . . . 1470
58.11.3 Property overview . . . 1470
58.11.4 TSimpleIPCClient.Create . . . 1470
58.11.5 TSimpleIPCClient.Destroy . . . 1470
58.11.6 TSimpleIPCClient.Connect . . . 1471
58.11.7 TSimpleIPCClient.Disconnect . . . 1471
58.11.8 TSimpleIPCClient.ServerRunning . . . 1471
58.11.9 TSimpleIPCClient.SendMessage . . . 1472
58.11.10 TSimpleIPCClient.SendStringMessage . . . 1472
58.11.11 TSimpleIPCClient.SendStringMessageFmt . . . 1472
58.11.12 TSimpleIPCClient.ServerInstance . . . 1472
58.12 TSimpleIPCServer . . . 1473
58.12.1 Description . . . 1473
58.12.2 Method overview . . . 1473
58.12.3 Property overview . . . 1473
58.12.4 TSimpleIPCServer.Create . . . 1474
58.12.5 TSimpleIPCServer.Destroy . . . 1474
58.12.6 TSimpleIPCServer.StartServer . . . 1474
58.12.7 TSimpleIPCServer.StopServer . . . 1475
58.12.8 TSimpleIPCServer.PeekMessage . . . 1475
58.12.9 TSimpleIPCServer.ReadMessage . . . 1475
58.12.10 TSimpleIPCServer.GetMessageData . . . 1475
58.12.11 TSimpleIPCServer.StringMessage . . . 1476
58.12.12 TSimpleIPCServer.Message . . . 1476

58.12.13TSimpleIPCServer.MsgType . . . 1476
58.12.14TSimpleIPCServer.MsgData . . . 1476
58.12.15TSimpleIPCServer.InstanceID . . . 1477
58.12.16TSimpleIPCServer.ThreadExecuting . . . 1477
58.12.17TSimpleIPCServer.ThreadError . . . 1477
58.12.18TSimpleIPCServer.Global . . . 1477
58.12.19TSimpleIPCServer.OnMessage . . . 1478
58.12.20TSimpleIPCServer.OnMessageQueued . . . 1478
58.12.21TSimpleIPCServer.OnMessageError . . . 1478
58.12.22TSimpleIPCServer.OnThreadError . . . 1478
58.12.23TSimpleIPCServer.MaxQueue . . . 1479
58.12.24TSimpleIPCServer.MaxAction . . . 1479
58.12.25TSimpleIPCServer.Threaded . . . 1479
58.12.26TSimpleIPCServer.ThreadTimeout . . . 1480
58.12.27TSimpleIPCServer.SynchronizeEvents . . . 1480

59 Reference for unit 'singleinstance' 1481

59.1 Used units . . . 1481
59.2 Overview . . . 1481
59.3 Constants, types and variables . . . 1481
59.3.1 Types . . . 1481
59.3.2 Variables . . . 1482
59.4 ESingleInstance . . . 1482
59.4.1 Description . . . 1482
59.5 TBaseSingleInstance . . . 1482
59.5.1 Description . . . 1482
59.5.2 Method overview . . . 1483
59.5.3 Property overview . . . 1483
59.5.4 TBaseSingleInstance.Create . . . 1483
59.5.5 TBaseSingleInstance.Destroy . . . 1483
59.5.6 TBaseSingleInstance.Start . . . 1483
59.5.7 TBaseSingleInstance.Stop . . . 1484
59.5.8 TBaseSingleInstance.ServerCheckMessages . . . 1484
59.5.9 TBaseSingleInstance.ClientPostParams . . . 1484
59.5.10 TBaseSingleInstance.TimeOutMessages . . . 1485
59.5.11 TBaseSingleInstance.TimeOutWaitForInstances . . . 1485
59.5.12 TBaseSingleInstance.OnServerReceivedParams . . . 1485
59.5.13 TBaseSingleInstance.StartResult . . . 1485
59.5.14 TBaseSingleInstance.IsServer . . . 1486
59.5.15 TBaseSingleInstance.IsClient . . . 1486

60 Reference for unit 'SQLDB' 1487
60.1 Used units . . . 1487
60.2 Overview . . . 1487
60.3 Using SQLDB to access databases. . . . 1488
60.4 Using the universal TSQLConnector type. . . . 1490
60.5 Retrieving Schema Information. . . . 1491
60.6 Automatic generation of update SQL statements. . . . 1491
60.7 Using parameters. . . . 1492
60.8 Constants, types and variables . . . 1493
60.8.1 Constants . . . 1493
60.8.2 Types . . . 1497
60.8.3 Variables . . . 1500
60.9 Procedures and functions . . . 1501
60.9.1 GetConnectionDef . . . 1501
60.9.2 GetConnectionList . . . 1501
60.9.3 RegisterConnection . . . 1501
60.9.4 UnRegisterConnection . . . 1502
60.10TSQLStatementInfo . . . 1502
60.11ESQLDatabaseError . . . 1502
60.11.1 Description . . . 1502
60.11.2 Method overview . . . 1502
60.11.3 ESQLDatabaseError.CreateFmt . . . 1502
60.12TConnectionDef . . . 1503
60.12.1 Description . . . 1503
60.12.2 Method overview . . . 1503
60.12.3 TConnectionDef.TypeName . . . 1503
60.12.4 TConnectionDef.ConnectionClass . . . 1504
60.12.5 TConnectionDef.Description . . . 1504
60.12.6 TConnectionDef.DefaultLibraryName . . . 1504
60.12.7 TConnectionDef.LoadFunction . . . 1504
60.12.8 TConnectionDef.UnLoadFunction . . . 1505
60.12.9 TConnectionDef.LoadedLibraryName . . . 1505
60.12.10TConnectionDef.ApplyParams . . . 1505
60.13TCustomSQLQuery . . . 1505
60.13.1 Description . . . 1505
60.13.2 Method overview . . . 1506
60.13.3 Property overview . . . 1506
60.13.4 TCustomSQLQuery.Create . . . 1506
60.13.5 TCustomSQLQuery.Destroy . . . 1506
60.13.6 TCustomSQLQuery.Prepare . . . 1507

60.13.7 TCustomSQLQuery.UnPrepare . . . 1507
60.13.8 TCustomSQLQuery.ExecSQL . . . 1507
60.13.9 TCustomSQLQuery.SetSchemaInfo . . . 1508
60.13.10TCustomSQLQuery.RowsAffected . . . 1508
60.13.11TCustomSQLQuery.ParamByName . . . 1508
60.13.12TCustomSQLQuery.MacroByName . . . 1509
60.13.13TCustomSQLQuery.ApplyUpdates . . . 1509
60.13.14TCustomSQLQuery.Post . . . 1509
60.13.15TCustomSQLQuery.Delete . . . 1509
60.13.16TCustomSQLQuery.Prepared . . . 1510
60.13.17TCustomSQLQuery.SQLConnection . . . 1510
60.13.18TCustomSQLQuery.SQLTransaction . . . 1510
60.14TCustomSQLStatement . . . 1510
60.14.1 Description . . . 1510
60.14.2 Method overview . . . 1511
60.14.3 Property overview . . . 1511
60.14.4 TCustomSQLStatement.Create . . . 1511
60.14.5 TCustomSQLStatement.Destroy . . . 1511
60.14.6 TCustomSQLStatement.Prepare . . . 1512
60.14.7 TCustomSQLStatement.Execute . . . 1512
60.14.8 TCustomSQLStatement.Unprepare . . . 1512
60.14.9 TCustomSQLStatement.ParamByName . . . 1513
60.14.10TCustomSQLStatement.RowsAffected . . . 1513
60.14.11TCustomSQLStatement.Prepared . . . 1513
60.15TServerIndexDefs . . . 1513
60.15.1 Description . . . 1513
60.15.2 Method overview . . . 1513
60.15.3 TServerIndexDefs.Create . . . 1514
60.15.4 TServerIndexDefs.Update . . . 1514
60.16TSQLConnection . . . 1514
60.16.1 Description . . . 1514
60.16.2 Method overview . . . 1514
60.16.3 Property overview . . . 1515
60.16.4 TSQLConnection.Create . . . 1515
60.16.5 TSQLConnection.Destroy . . . 1515
60.16.6 TSQLConnection.StartTransaction . . . 1516
60.16.7 TSQLConnection.EndTransaction . . . 1516
60.16.8 TSQLConnection.ExecuteDirect . . . 1516
60.16.9 TSQLConnection.GetObjectNames . . . 1517
60.16.10TSQLConnection.HasTable . . . 1517

60.16.11TSQLConnection.GetTableNames . . . 1517
60.16.12TSQLConnection.GetProcedureNames . . . 1517
60.16.13TSQLConnection.GetFieldNames . . . 1518
60.16.14TSQLConnection.GetSchemaNames . . . 1518
60.16.15TSQLConnection.GetSequenceNames . . . 1518
60.16.16TSQLConnection.GetConnectionInfo . . . 1518
60.16.17TSQLConnection.GetStatementInfo . . . 1519
60.16.18TSQLConnection.CreateDB . . . 1519
60.16.19TSQLConnection.DropDB . . . 1519
60.16.20TSQLConnection.GetNextValue . . . 1519
60.16.21TSQLConnection.ConnOptions . . . 1520
60.16.22TSQLConnection.Handle . . . 1520
60.16.23TSQLConnection.FieldNameQuoteChars . . . 1520
60.16.24TSQLConnection.Password . . . 1521
60.16.25TSQLConnection.Transaction . . . 1521
60.16.26TSQLConnection.UserName . . . 1521
60.16.27TSQLConnection.CharSet . . . 1522
60.16.28TSQLConnection.HostName . . . 1522
60.16.29TSQLConnection.OnLog . . . 1522
60.16.30TSQLConnection.LogEvents . . . 1523
60.16.31TSQLConnection.Options . . . 1523
60.16.32TSQLConnection.Role . . . 1523
60.16.33TSQLConnection.Connected . . . 1524
60.16.34TSQLConnection.DatabaseName . . . 1524
60.16.35TSQLConnection.KeepConnection . . . 1524
60.16.36TSQLConnection.LoginPrompt . . . 1525
60.16.37TSQLConnection.Params . . . 1525
60.16.38TSQLConnection.OnLogin . . . 1525
60.17TSQLConnector . . . 1525
60.17.1 Description . . . 1525
60.17.2 Property overview . . . 1526
60.17.3 TSQLConnector.ConnectorType . . . 1526
60.17.4 TSQLConnector.Port . . . 1526
60.18TSQLCursor . . . 1526
60.18.1 Description . . . 1526
60.19TSQLDBFieldDef . . . 1527
60.19.1 Description . . . 1527
60.19.2 Property overview . . . 1527
60.19.3 TSQLDBFieldDef.SQLDBData . . . 1527
60.20TSQLDBFieldDefs . . . 1527

60.20.1 Description . . . 1527
60.21 TSQLDBParam . . . 1527
60.21.1 Description . . . 1527
60.21.2 Property overview . . . 1527
60.21.3 TSQLDBParam.FieldDef . . . 1528
60.21.4 TSQLDBParam.SQLDBData . . . 1528
60.22 TSQLDBParams . . . 1528
60.22.1 Description . . . 1528
60.23 TSQLHandle . . . 1528
60.23.1 Description . . . 1528
60.24 TSQLQuery . . . 1529
60.24.1 Description . . . 1529
60.24.2 Property overview . . . 1531
60.24.3 TSQLQuery.SchemaType . . . 1532
60.24.4 TSQLQuery.StatementType . . . 1532
60.24.5 TSQLQuery.MaxIndexesCount . . . 1532
60.24.6 TSQLQuery.FieldDefs . . . 1532
60.24.7 TSQLQuery.Active . . . 1533
60.24.8 TSQLQuery.AutoCalcFields . . . 1533
60.24.9 TSQLQuery.Filter . . . 1533
60.24.10 TSQLQuery.Filtered . . . 1533
60.24.11 TSQLQuery.AfterCancel . . . 1533
60.24.12 TSQLQuery.AfterClose . . . 1533
60.24.13 TSQLQuery.AfterDelete . . . 1533
60.24.14 TSQLQuery.AfterEdit . . . 1534
60.24.15 TSQLQuery.AfterInsert . . . 1534
60.24.16 TSQLQuery.AfterOpen . . . 1534
60.24.17 TSQLQuery.AfterPost . . . 1534
60.24.18 TSQLQuery.AfterRefresh . . . 1534
60.24.19 TSQLQuery.AfterScroll . . . 1534
60.24.20 TSQLQuery.BeforeCancel . . . 1534
60.24.21 TSQLQuery.BeforeClose . . . 1535
60.24.22 TSQLQuery.BeforeDelete . . . 1535
60.24.23 TSQLQuery.BeforeEdit . . . 1535
60.24.24 TSQLQuery.BeforeInsert . . . 1535
60.24.25 TSQLQuery.BeforeOpen . . . 1535
60.24.26 TSQLQuery.BeforePost . . . 1535
60.24.27 TSQLQuery.BeforeRefresh . . . 1535
60.24.28 TSQLQuery.BeforeScroll . . . 1536
60.24.29 TSQLQuery.OnCalcFields . . . 1536

60.24.30TSQLQuery.OnDeleteError . . . 1536
60.24.31TSQLQuery.OnEditError . . . 1536
60.24.32TSQLQuery.OnFilterRecord . . . 1536
60.24.33TSQLQuery.OnNewRecord . . . 1536
60.24.34TSQLQuery.OnPostError . . . 1536
60.24.35TSQLQuery.Database . . . 1537
60.24.36TSQLQuery.Transaction . . . 1537
60.24.37TSQLQuery.ReadOnly . . . 1537
60.24.38TSQLQuery.SQL . . . 1537
60.24.39TSQLQuery.InsertSQL . . . 1538
60.24.40TSQLQuery.UpdateSQL . . . 1538
60.24.41TSQLQuery.DeleteSQL . . . 1539
60.24.42TSQLQuery.RefreshSQL . . . 1539
60.24.43TSQLQuery.IndexDefs . . . 1539
60.24.44TSQLQuery.Options . . . 1540
60.24.45TSQLQuery.Params . . . 1540
60.24.46TSQLQuery.ParamCheck . . . 1541
60.24.47TSQLQuery.Macros . . . 1541
60.24.48TSQLQuery.MacroCheck . . . 1541
60.24.49TSQLQuery.MacroChar . . . 1541
60.24.50TSQLQuery.ParseSQL . . . 1542
60.24.51TSQLQuery.UpdateMode . . . 1542
60.24.52TSQLQuery.UsePrimaryKeyAsKey . . . 1542
60.24.53TSQLQuery.DataSource . . . 1543
60.24.54TSQLQuery.Sequence . . . 1543
60.24.55TSQLQuery.ServerFilter . . . 1544
60.24.56TSQLQuery.ServerFiltered . . . 1544
60.24.57TSQLQuery.ServerIndexDefs . . . 1544
60.25TSQLScript . . . 1545
60.25.1 Description . . . 1545
60.25.2 Method overview . . . 1545
60.25.3 Property overview . . . 1545
60.25.4 TSQLScript.Create . . . 1545
60.25.5 TSQLScript.Destroy . . . 1546
60.25.6 TSQLScript.Execute . . . 1546
60.25.7 TSQLScript.ExecuteScript . . . 1546
60.25.8 TSQLScript.Aborted . . . 1546
60.25.9 TSQLScript.Line . . . 1547
60.25.10TSQLScript.DataBase . . . 1547
60.25.11TSQLScript.Transaction . . . 1547

60.25.12TSQLScript.OnDirective . . . 1547
60.25.13TSQLScript.AutoCommit . . . 1548
60.25.14TSQLScript.UseDollarString . . . 1548
60.25.15TSQLScript.DollarStrings . . . 1548
60.25.16TSQLScript.Directives . . . 1549
60.25.17TSQLScript.Defines . . . 1549
60.25.18TSQLScript.Script . . . 1549
60.25.19TSQLScript.Terminator . . . 1550
60.25.20TSQLScript.CommentsinSQL . . . 1550
60.25.21TSQLScript.UseSetTerm . . . 1550
60.25.22TSQLScript.UseCommit . . . 1551
60.25.23TSQLScript.UseDefines . . . 1551
60.25.24TSQLScript.OnException . . . 1552
60.26TSQLSequence . . . 1552
60.26.1 Description . . . 1552
60.26.2 Method overview . . . 1552
60.26.3 Property overview . . . 1552
60.26.4 TSQLSequence.Create . . . 1552
60.26.5 TSQLSequence.Assign . . . 1553
60.26.6 TSQLSequence.Apply . . . 1553
60.26.7 TSQLSequence.GetNextValue . . . 1553
60.26.8 TSQLSequence.FieldName . . . 1553
60.26.9 TSQLSequence.SequenceName . . . 1554
60.26.10TSQLSequence.IncrementBy . . . 1554
60.26.11TSQLSequence.ApplyEvent . . . 1554
60.27TSQLStatement . . . 1554
60.27.1 Description . . . 1554
60.27.2 Property overview . . . 1555
60.27.3 TSQLStatement.Database . . . 1555
60.27.4 TSQLStatement.DataSource . . . 1555
60.27.5 TSQLStatement.ParamCheck . . . 1555
60.27.6 TSQLStatement.Params . . . 1556
60.27.7 TSQLStatement.MacroCheck . . . 1556
60.27.8 TSQLStatement.Macros . . . 1556
60.27.9 TSQLStatement.ParseSQL . . . 1557
60.27.10TSQLStatement.SQL . . . 1557
60.27.11TSQLStatement.Transaction . . . 1557
60.28TSQLTransaction . . . 1558
60.28.1 Description . . . 1558
60.28.2 Method overview . . . 1558

60.28.3 Property overview . . . 1558
60.28.4 TSQLTransaction.Create . . . 1558
60.28.5 TSQLTransaction.Destroy . . . 1558
60.28.6 TSQLTransaction.Commit . . . 1559
60.28.7 TSQLTransaction.CommitRetaining . . . 1559
60.28.8 TSQLTransaction.Rollback . . . 1559
60.28.9 TSQLTransaction.RollbackRetaining . . . 1560
60.28.10 TSQLTransaction.StartTransaction . . . 1560
60.28.11 TSQLTransaction.EndTransaction . . . 1560
60.28.12 TSQLTransaction.Handle . . . 1561
60.28.13 TSQLTransaction.SQLConnection . . . 1561
60.28.14 TSQLTransaction.Action . . . 1561
60.28.15 TSQLTransaction.Database . . . 1561
60.28.16 TSQLTransaction.Params . . . 1562
60.28.17 TSQLTransaction.Options . . . 1562

61 Reference for unit 'SQLTypes' 1563
61.1 Used units . . . 1563
61.2 Constants, types and variables . . . 1563
61.2.1 Types . . . 1563
61.3 TSqlObjectIdenfier . . . 1565
61.3.1 Description . . . 1565
61.3.2 Method overview . . . 1565
61.3.3 Property overview . . . 1565
61.3.4 TSqlObjectIdenfier.Create . . . 1565
61.3.5 TSqlObjectIdenfier.FullName . . . 1566
61.3.6 TSqlObjectIdenfier.SchemaName . . . 1566
61.3.7 TSqlObjectIdenfier.ObjectName . . . 1566
61.4 TSqlObjectIdentifierList . . . 1566
61.4.1 Method overview . . . 1566
61.4.2 Property overview . . . 1567
61.4.3 TSqlObjectIdentifierList.AddIdentifier . . . 1567
61.4.4 TSqlObjectIdentifierList.Identifiers . . . 1567

62 Reference for unit 'streamcoll' 1568
62.1 Used units . . . 1568
62.2 Overview . . . 1568
62.3 Procedures and functions . . . 1568
62.3.1 ColReadBoolean . . . 1568
62.3.2 ColReadCurrency . . . 1569
62.3.3 ColReadDateTime . . . 1569

62.3.4 ColReadFloat . . . 1569
62.3.5 ColReadInteger . . . 1569
62.3.6 ColReadString . . . 1570
62.3.7 ColWriteBoolean . . . 1570
62.3.8 ColWriteCurrency . . . 1570
62.3.9 ColWriteDateTime . . . 1570
62.3.10 ColWriteFloat . . . 1571
62.3.11 ColWriteInteger . . . 1571
62.3.12 ColWriteString . . . 1571
62.4 EStreamColl . . . 1571
62.4.1 Description . . . 1571
62.5 TStreamCollection . . . 1571
62.5.1 Description . . . 1571
62.5.2 Method overview . . . 1572
62.5.3 Property overview . . . 1572
62.5.4 TStreamCollection.LoadFromStream . . . 1572
62.5.5 TStreamCollection.SaveToStream . . . 1572
62.5.6 TStreamCollection.Streaming . . . 1572
62.6 TStreamCollectionItem . . . 1573
62.6.1 Description . . . 1573

63 Reference for unit 'streamex' 1574
63.1 Used units . . . 1574
63.2 Overview . . . 1574
63.3 Constants, types and variables . . . 1574
63.3.1 Constants . . . 1574
63.4 TBidirBinaryObjectReader . . . 1575
63.4.1 Description . . . 1575
63.4.2 Property overview . . . 1575
63.4.3 TBidirBinaryObjectReader.Position . . . 1575
63.5 TBidirBinaryObjectWriter . . . 1575
63.5.1 Description . . . 1575
63.5.2 Property overview . . . 1575
63.5.3 TBidirBinaryObjectWriter.Position . . . 1575
63.6 TDelphiReader . . . 1576
63.6.1 Description . . . 1576
63.6.2 Method overview . . . 1576
63.6.3 Property overview . . . 1576
63.6.4 TDelphiReader.GetDriver . . . 1576
63.6.5 TDelphiReader.ReadStr . . . 1576

63.6.6 TDelphiReader.Read . . . 1577
63.6.7 TDelphiReader.Position . . . 1577
63.7 TDelphiWriter . . . 1577
63.7.1 Description . . . 1577
63.7.2 Method overview . . . 1577
63.7.3 Property overview . . . 1577
63.7.4 TDelphiWriter.GetDriver . . . 1578
63.7.5 TDelphiWriter.FlushBuffer . . . 1578
63.7.6 TDelphiWriter.Write . . . 1578
63.7.7 TDelphiWriter.WriteStr . . . 1578
63.7.8 TDelphiWriter.WriteValue . . . 1578
63.7.9 TDelphiWriter.Position . . . 1579
63.8 TFileReader . . . 1579
63.8.1 Description . . . 1579
63.8.2 Method overview . . . 1579
63.8.3 TFileReader.Create . . . 1579
63.8.4 TFileReader.Destroy . . . 1580
63.8.5 TFileReader.Reset . . . 1580
63.8.6 TFileReader.Close . . . 1580
63.8.7 TFileReader.ReadLine . . . 1580
63.9 TStreamHelper . . . 1581
63.9.1 Description . . . 1581
63.9.2 Method overview . . . 1581
63.9.3 TStreamHelper.ReadWordLE . . . 1581
63.9.4 TStreamHelper.ReadDWordLE . . . 1581
63.9.5 TStreamHelper.ReadQWordLE . . . 1582
63.9.6 TStreamHelper.WriteWordLE . . . 1582
63.9.7 TStreamHelper.WriteDWordLE . . . 1582
63.9.8 TStreamHelper.WriteQWordLE . . . 1582
63.9.9 TStreamHelper.ReadWordBE . . . 1583
63.9.10 TStreamHelper.ReadDWordBE . . . 1583
63.9.11 TStreamHelper.ReadQWordBE . . . 1583
63.9.12 TStreamHelper.WriteWordBE . . . 1583
63.9.13 TStreamHelper.WriteDWordBE . . . 1584
63.9.14 TStreamHelper.WriteQWordBE . . . 1584
63.9.15 TStreamHelper.ReadSingle . . . 1584
63.9.16 TStreamHelper.ReadDouble . . . 1584
63.9.17 TStreamHelper.WriteSingle . . . 1585
63.9.18 TStreamHelper.WriteDouble . . . 1585
63.10TStreamReader . . . 1585

63.10.1 Description . . . 1585
63.10.2 Method overview . . . 1585
63.10.3 Property overview . . . 1585
63.10.4 TStreamReader.Create . . . 1586
63.10.5 TStreamReader.Destroy . . . 1586
63.10.6 TStreamReader.Reset . . . 1586
63.10.7 TStreamReader.Close . . . 1587
63.10.8 TStreamReader.ReadLine . . . 1587
63.10.9 TStreamReader.BaseStream . . . 1587
63.10.10TStreamReader.OwnsStream . . . 1587
63.11TStreamWriter . . . 1588
63.11.1 Method overview . . . 1588
63.11.2 Property overview . . . 1588
63.11.3 TStreamWriter.Create . . . 1588
63.11.4 TStreamWriter.Destroy . . . 1588
63.11.5 TStreamWriter.Close . . . 1588
63.11.6 TStreamWriter.Flush . . . 1588
63.11.7 TStreamWriter.OwnStream . . . 1589
63.11.8 TStreamWriter.Write . . . 1589
63.11.9 TStreamWriter.WriteLine . . . 1589
63.11.10TStreamWriter.AutoFlush . . . 1589
63.11.11TStreamWriter.NewLine . . . 1590
63.11.12TStreamWriter.Encoding . . . 1590
63.11.13TStreamWriter.BaseStream . . . 1590
63.12TStringReader . . . 1590
63.12.1 Description . . . 1590
63.12.2 Method overview . . . 1590
63.12.3 TStringReader.Create . . . 1590
63.12.4 TStringReader.Destroy . . . 1591
63.12.5 TStringReader.Reset . . . 1591
63.12.6 TStringReader.Close . . . 1591
63.12.7 TStringReader.ReadLine . . . 1591
63.13TStringWriter . . . 1592
63.13.1 Method overview . . . 1592
63.13.2 TStringWriter.Create . . . 1592
63.13.3 TStringWriter.Destroy . . . 1592
63.13.4 TStringWriter.Close . . . 1592
63.13.5 TStringWriter.Flush . . . 1592
63.13.6 TStringWriter.Write . . . 1592
63.13.7 TStringWriter.WriteLine . . . 1593

63.13.8 TStringWriter.ToString . . . 1593
63.14TTextReader . . . 1593
63.14.1 Description . . . 1593
63.14.2 Method overview . . . 1593
63.14.3 Property overview . . . 1594
63.14.4 TTextReader.Create . . . 1594
63.14.5 TTextReader.Reset . . . 1594
63.14.6 TTextReader.Close . . . 1594
63.14.7 TTextReader.ReadLine . . . 1594
63.14.8 TTextReader.Eof . . . 1595
63.14.9 TTextReader.EndOfStream . . . 1595
63.15TTextWriter . . . 1595
63.15.1 Method overview . . . 1595
63.15.2 TTextWriter.Close . . . 1595
63.15.3 TTextWriter.Flush . . . 1595
63.15.4 TTextWriter.Write . . . 1595
63.15.5 TTextWriter.WriteLine . . . 1596
63.16TWindowedStream . . . 1596
63.16.1 Description . . . 1596
63.16.2 Method overview . . . 1597
63.16.3 TWindowedStream.Create . . . 1597
63.16.4 TWindowedStream.Destroy . . . 1597
63.16.5 TWindowedStream.Read . . . 1597
63.16.6 TWindowedStream.Write . . . 1598
63.16.7 TWindowedStream.Seek . . . 1598

64 Reference for unit 'StreamIO' 1599
64.1 Used units . . . 1599
64.2 Overview . . . 1599
64.3 Procedures and functions . . . 1599
64.3.1 AssignStream . . . 1599
64.3.2 GetStream . . . 1600

65 Reference for unit 'syncobjs' 1601
65.1 Used units . . . 1601
65.2 Overview . . . 1601
65.3 Constants, types and variables . . . 1601
65.3.1 Constants . . . 1601
65.3.2 Types . . . 1601
65.4 ELockException . . . 1602
65.4.1 Description . . . 1602

65.5 ELockRecursionException . . . 1602
65.5.1 Description . . . 1602
65.6 ESyncObjectException . . . 1602
65.6.1 Description . . . 1602
65.7 TCriticalSection . . . 1603
65.7.1 Description . . . 1603
65.7.2 Method overview . . . 1603
65.7.3 TCriticalSection.Acquire . . . 1603
65.7.4 TCriticalSection.Release . . . 1603
65.7.5 TCriticalSection.Enter . . . 1604
65.7.6 TCriticalSection.TryEnter . . . 1604
65.7.7 TCriticalSection.Leave . . . 1604
65.7.8 TCriticalSection.Create . . . 1605
65.7.9 TCriticalSection.Destroy . . . 1605
65.8 TEventObject . . . 1605
65.8.1 Description . . . 1605
65.8.2 Method overview . . . 1605
65.8.3 Property overview . . . 1605
65.8.4 TEventObject.Create . . . 1606
65.8.5 TEventObject.ResetEvent . . . 1606
65.8.6 TEventObject.SetEvent . . . 1606
65.8.7 TEventObject.ManualReset . . . 1606
65.9 THandleObject . . . 1607
65.9.1 Description . . . 1607
65.9.2 Method overview . . . 1607
65.9.3 Property overview . . . 1607
65.9.4 THandleObject.Create . . . 1607
65.9.5 THandleObject.Destroy . . . 1607
65.9.6 THandleObject.WaitFor . . . 1607
65.9.7 THandleObject.Handle . . . 1607
65.9.8 THandleObject.LastError . . . 1608
65.10 TSimpleEvent . . . 1608
65.10.1 Description . . . 1608
65.10.2 Method overview . . . 1608
65.10.3 TSimpleEvent.Create . . . 1608
65.11 TSynchroObject . . . 1609
65.11.1 Description . . . 1609
65.11.2 Method overview . . . 1609
65.11.3 TSynchroObject.Acquire . . . 1609
65.11.4 TSynchroObject.Release . . . 1609

66 Reference for unit 'URIParser' 1610
66.1 Used units . . . 1610
66.2 Overview . . . 1610
66.3 Basic usage of URI parsing functions . . . 1610
66.4 Constants, types and variables . . . 1613
66.4.1 Types . . . 1613
66.5 Procedures and functions . . . 1613
66.5.1 EncodeURI . . . 1613
66.5.2 FilenameToURI . . . 1614
66.5.3 IsAbsoluteURI . . . 1614
66.5.4 ParseURI . . . 1614
66.5.5 ResolveRelativeURI . . . 1614
66.5.6 URIToFilename . . . 1615
66.6 TURI . . . 1615
66.6.1 Method overview . . . 1616
66.6.2 TURI.create . . . 1616
66.6.3 TURI.ToString . . . 1616
66.6.4 TURI.URI . . . 1616
66.6.5 TURI.Free . . . 1616

67 Reference for unit 'wformat' 1617
67.1 Used units . . . 1617
67.2 Overview . . . 1617
67.3 Constants, types and variables . . . 1617
67.3.1 Constants . . . 1617
67.3.2 Types . . . 1617
67.4 TFormattingWriter . . . 1618
67.4.1 Description . . . 1618
67.4.2 Method overview . . . 1619
67.4.3 TFormattingWriter.Create . . . 1619
67.4.4 TFormattingWriter.EscapeText . . . 1620
67.4.5 TFormattingWriter.Dump . . . 1620
67.4.6 TFormattingWriter.DumpLn . . . 1620
67.4.7 TFormattingWriter.Write . . . 1620
67.4.8 TFormattingWriter.WriteFmt . . . 1621
67.4.9 TFormattingWriter.DocumentStart . . . 1621
67.4.10 TFormattingWriter.DocumentEnd . . . 1621
67.4.11 TFormattingWriter.Header . . . 1621
67.4.12 TFormattingWriter.HeaderStart . . . 1622
67.4.13 TFormattingWriter.HeaderEnd . . . 1622

67.4.14 TFormattingWriter.ParagraphStart . . . 1622
67.4.15 TFormattingWriter.ParagraphEnd . . . 1622
67.4.16 TFormattingWriter.LineBreak . . . 1622
67.4.17 TFormattingWriter.Rule . . . 1623
67.4.18 TFormattingWriter.BoldStart . . . 1623
67.4.19 TFormattingWriter.BoldEnd . . . 1623
67.4.20 TFormattingWriter.ItalicStart . . . 1623
67.4.21 TFormattingWriter.ItalicEnd . . . 1623
67.4.22 TFormattingWriter.UnderlineStart . . . 1624
67.4.23 TFormattingWriter.UnderlineEnd . . . 1624
67.4.24 TFormattingWriter.PreformatStart . . . 1624
67.4.25 TFormattingWriter.PreformatEnd . . . 1624
67.4.26 TFormattingWriter.TableStart . . . 1625
67.4.27 TFormattingWriter.TableEnd . . . 1625
67.4.28 TFormattingWriter.RowStart . . . 1625
67.4.29 TFormattingWriter.RowEnd . . . 1625
67.4.30 TFormattingWriter.RowNext . . . 1626
67.4.31 TFormattingWriter.CellStart . . . 1626
67.4.32 TFormattingWriter.CellEnd . . . 1626
67.4.33 TFormattingWriter.CellNext . . . 1626
67.4.34 TFormattingWriter.HeaderCellStart . . . 1627
67.4.35 TFormattingWriter.HeaderCellEnd . . . 1627
67.4.36 TFormattingWriter.HeaderCellNext . . . 1627
67.4.37 TFormattingWriter.ListStart . . . 1627
67.4.38 TFormattingWriter.ListEnd . . . 1628
67.4.39 TFormattingWriter.ListItemStart . . . 1628
67.4.40 TFormattingWriter.ListItemEnd . . . 1628
67.4.41 TFormattingWriter.ListItem . . . 1628
67.4.42 TFormattingWriter.DefinitionItem . . . 1629
67.4.43 TFormattingWriter.WriteList . . . 1629

68 Reference for unit 'whtml' 1630

68.1 Used units . . . 1630
68.2 Overview . . . 1630
68.3 Constants, types and variables . . . 1630
68.3.1 Constants . . . 1630
68.4 THTMLWriter . . . 1630
68.4.1 Description . . . 1630
68.4.2 Method overview . . . 1631
68.4.3 THTMLWriter.Create . . . 1632

68.4.4 THTMLWriter.TagStart . . . 1632
68.4.5 THTMLWriter.TagEnd . . . 1632
68.4.6 THTMLWriter.EscapeText . . . 1632
68.4.7 THTMLWriter.DocumentStart . . . 1633
68.4.8 THTMLWriter.DocumentEnd . . . 1633
68.4.9 THTMLWriter.HeaderStart . . . 1633
68.4.10 THTMLWriter.HeaderEnd . . . 1633
68.4.11 THTMLWriter.ParagraphStart . . . 1633
68.4.12 THTMLWriter.ParagraphEnd . . . 1634
68.4.13 THTMLWriter.LineBreak . . . 1634
68.4.14 THTMLWriter.Rule . . . 1634
68.4.15 THTMLWriter.BoldStart . . . 1634
68.4.16 THTMLWriter.BoldEnd . . . 1634
68.4.17 THTMLWriter.ItalicStart . . . 1635
68.4.18 THTMLWriter.ItalicEnd . . . 1635
68.4.19 THTMLWriter.UnderlineStart . . . 1635
68.4.20 THTMLWriter.UnderlineEnd . . . 1635
68.4.21 THTMLWriter.PreformatStart . . . 1635
68.4.22 THTMLWriter.PreformatEnd . . . 1636
68.4.23 THTMLWriter.TableStart . . . 1636
68.4.24 THTMLWriter.TableEnd . . . 1636
68.4.25 THTMLWriter.RowStart . . . 1636
68.4.26 THTMLWriter.RowEnd . . . 1636
68.4.27 THTMLWriter.CellStart . . . 1637
68.4.28 THTMLWriter.CellEnd . . . 1637
68.4.29 THTMLWriter.HeaderCellStart . . . 1637
68.4.30 THTMLWriter.HeaderCellEnd . . . 1637
68.4.31 THTMLWriter.ListStart . . . 1637
68.4.32 THTMLWriter.ListEnd . . . 1638
68.4.33 THTMLWriter.ListItemStart . . . 1638
68.4.34 THTMLWriter.ListItemEnd . . . 1638
68.4.35 THTMLWriter.DefinitionItem . . . 1638
68.4.36 THTMLWriter.FormStart . . . 1639
68.4.37 THTMLWriter.FormEnd . . . 1639
68.4.38 THTMLWriter.EmitInput . . . 1639
68.4.39 THTMLWriter.EmitPasswordInput . . . 1639
68.4.40 THTMLWriter.EmitCheckBox . . . 1640
68.4.41 THTMLWriter.EmitRadioButton . . . 1640
68.4.42 THTMLWriter.EmitArea . . . 1640
68.4.43 THTMLWriter.EmitComboBox . . . 1640

68.4.44 THTMLWriter.EmitButton 1641
68.4.45 THTMLWriter.EmitSubmitButton 1641
68.4.46 THTMLWriter.EmitResetButton 1641
68.4.47 THTMLWriter.EmitHiddenVar 1641

69 Reference for unit 'wtex' 1642
69.1 Used units 1642
69.2 Overview 1642
69.3 Constants, types and variables 1642
69.3.1 Constants 1642
69.4 TTexWriter 1643
69.4.1 Description 1643
69.4.2 Method overview 1643
69.4.3 TTexWriter.ScopeStart 1643
69.4.4 TTexWriter.ScopeEnd 1644
69.4.5 TTexWriter.EnvironmentStart 1644
69.4.6 TTexWriter.EnvironmentEnd 1644
69.4.7 TTexWriter.EscapeText 1644
69.4.8 TTexWriter.DocumentStart 1645
69.4.9 TTexWriter.DocumentEnd 1645
69.4.10 TTexWriter.HeaderStart 1645
69.4.11 TTexWriter.HeaderEnd 1645
69.4.12 TTexWriter.ParagraphEnd 1645
69.4.13 TTexWriter.LineBreak 1646
69.4.14 TTexWriter.Rule 1646
69.4.15 TTexWriter.BoldStart 1646
69.4.16 TTexWriter.BoldEnd 1646
69.4.17 TTexWriter.ItalicStart 1646
69.4.18 TTexWriter.ItalicEnd 1647
69.4.19 TTexWriter.UnderlineStart 1647
69.4.20 TTexWriter.UnderlineEnd 1647
69.4.21 TTexWriter.PreformatStart 1647
69.4.22 TTexWriter.PreformatEnd 1647
69.4.23 TTexWriter.TableStart 1648
69.4.24 TTexWriter.TableEnd 1648
69.4.25 TTexWriter.RowStart 1648
69.4.26 TTexWriter.RowEnd 1648
69.4.27 TTexWriter.CellStart 1648
69.4.28 TTexWriter.HeaderCellStart 1649
69.4.29 TTexWriter.HeaderCellEnd 1649

69.4.30 TTexWriter.ListStart . . . 1649
69.4.31 TTexWriter.ListEnd . . . 1649
69.4.32 TTexWriter.ListItemStart . . . 1650
69.4.33 TTexWriter.DefinitionItem . . . 1650

70 Reference for unit 'Zipper' 1651
70.1 Used units . . . 1651
70.2 Overview . . . 1651
70.3 Constants, types and variables . . . 1651
70.3.1 Constants . . . 1651
70.3.2 Types . . . 1655
70.4 Central_File_Header_Type . . . 1656
70.5 CodeRec . . . 1657
70.6 End_of_Central_Dir_Type . . . 1657
70.7 Extensible_Data_Field_Header_Type . . . 1657
70.8 Local_File_Header_Type . . . 1657
70.9 Zip64_End_of_Central_Dir_Locator_type . . . 1658
70.10 Zip64_End_of_Central_Dir_type . . . 1658
70.11 Zip64_Extended_Info_Field_Type . . . 1659
70.12 EZipError . . . 1659
70.12.1 Description . . . 1659
70.13 TCompressor . . . 1659
70.13.1 Description . . . 1659
70.13.2 Method overview . . . 1659
70.13.3 Property overview . . . 1659
70.13.4 TCompressor.Create . . . 1659
70.13.5 TCompressor.Compress . . . 1660
70.13.6 TCompressor.ZipID . . . 1660
70.13.7 TCompressor.ZipVersionReqd . . . 1660
70.13.8 TCompressor.ZipBitFlag . . . 1660
70.13.9 TCompressor.Terminate . . . 1660
70.13.10 TCompressor.BufferSize . . . 1660
70.13.11 TCompressor.OnPercent . . . 1661
70.13.12 TCompressor.OnProgress . . . 1661
70.13.13 TCompressor.Crc32Val . . . 1661
70.13.14 TCompressor.Terminated . . . 1661
70.14 TDeCompressor . . . 1661
70.14.1 Description . . . 1661
70.14.2 Method overview . . . 1662
70.14.3 Property overview . . . 1662

70.14.4 TDeCompressor.Create 1662
70.14.5 TDeCompressor.DeCompress 1662
70.14.6 TDeCompressor.Terminate 1662
70.14.7 TDeCompressor.ZipID 1662
70.14.8 TDeCompressor.BufferSize 1663
70.14.9 TDeCompressor.OnPercent 1663
70.14.10TDeCompressor.OnProgress 1663
70.14.11TDeCompressor.OnProgressEx 1663
70.14.12TDeCompressor.Crc32Val 1663
70.14.13TDeCompressor.Terminated 1664
70.15TDeflater 1664
70.15.1 Description 1664
70.15.2 Method overview 1664
70.15.3 Property overview 1664
70.15.4 TDeflater.Create 1664
70.15.5 TDeflater.Compress 1664
70.15.6 TDeflater.ZipID 1665
70.15.7 TDeflater.ZipVersionReqd 1665
70.15.8 TDeflater.ZipBitFlag 1665
70.15.9 TDeflater.CompressionLevel 1665
70.16TFullZipFileEntries 1666
70.16.1 Description 1666
70.16.2 Property overview 1666
70.16.3 TFullZipFileEntries.FullEntries 1666
70.17TFullZipFileEntry 1666
70.17.1 Description 1666
70.17.2 Property overview 1666
70.17.3 TFullZipFileEntry.BitFlags 1667
70.17.4 TFullZipFileEntry.CompressMethod 1667
70.17.5 TFullZipFileEntry.CompressedSize 1667
70.17.6 TFullZipFileEntry.CRC32 1667
70.18TInflater 1667
70.18.1 Description 1667
70.18.2 Method overview 1668
70.18.3 TInflater.Create 1668
70.18.4 TInflater.DeCompress 1668
70.18.5 TInflater.ZipID 1668
70.19TShrinker 1668
70.19.1 Description 1668
70.19.2 Method overview 1668

70.19.3 TShrinker.Create . . . 1669
70.19.4 TShrinker.Destroy . . . 1669
70.19.5 TShrinker.Compress . . . 1669
70.19.6 TShrinker.ZipID . . . 1669
70.19.7 TShrinker.ZipVersionReqd . . . 1669
70.19.8 TShrinker.ZipBitFlag . . . 1669
70.20 TUnZipper . . . 1670
70.20.1 Description . . . 1670
70.20.2 Method overview . . . 1670
70.20.3 Property overview . . . 1670
70.20.4 TUnZipper.Create . . . 1671
70.20.5 TUnZipper.Destroy . . . 1671
70.20.6 TUnZipper.UnZipOneFile . . . 1671
70.20.7 TUnZipper.UnZipAllFiles . . . 1671
70.20.8 TUnZipper.UnZipFile . . . 1672
70.20.9 TUnZipper.UnZipFiles . . . 1672
70.20.10 TUnZipper.Unzip . . . 1672
70.20.11 TUnZipper.Clear . . . 1673
70.20.12 TUnZipper.Examine . . . 1673
70.20.13 TUnZipper.Terminate . . . 1673
70.20.14 TUnZipper.BufferSize . . . 1673
70.20.15 TUnZipper.OnOpenInputStream . . . 1673
70.20.16 TUnZipper.OnCloseInputStream . . . 1674
70.20.17 TUnZipper.OnCreateStream . . . 1674
70.20.18 TUnZipper.OnDoneStream . . . 1674
70.20.19 TUnZipper.OnPercent . . . 1674
70.20.20 TUnZipper.OnProgress . . . 1674
70.20.21 TUnZipper.OnProgressEx . . . 1675
70.20.22 TUnZipper.OnStartFile . . . 1675
70.20.23 TUnZipper.OnEndFile . . . 1675
70.20.24 TUnZipper.FileName . . . 1675
70.20.25 TUnZipper.OutputPath . . . 1675
70.20.26 TUnZipper.FileComment . . . 1676
70.20.27 TUnZipper.Files . . . 1676
70.20.28 TUnZipper.Entries . . . 1676
70.20.29 TUnZipper.UseUTF8 . . . 1676
70.20.30 TUnZipper.Flat . . . 1677
70.20.31 TUnZipper.Terminated . . . 1677
70.21 TZipFileEntries . . . 1677
70.21.1 Description . . . 1677

70.21.2 Method overview . . . 1677
70.21.3 Property overview . . . 1678
70.21.4 TZipFileEntries.AddFileEntry . . . 1678
70.21.5 TZipFileEntries.AddFileEntries . . . 1678
70.21.6 TZipFileEntries.Entries . . . 1678
70.22 TZipFileEntry . . . 1679
70.22.1 Description . . . 1679
70.22.2 Method overview . . . 1679
70.22.3 Property overview . . . 1679
70.22.4 TZipFileEntry.Create . . . 1680
70.22.5 TZipFileEntry.IsDirectory . . . 1680
70.22.6 TZipFileEntry.IsLink . . . 1680
70.22.7 TZipFileEntry.Assign . . . 1680
70.22.8 TZipFileEntry.Stream . . . 1680
70.22.9 TZipFileEntry.ArchiveFileName . . . 1681
70.22.10 TZipFileEntry.UTF8ArchiveFileName . . . 1681
70.22.11 TZipFileEntry.DiskFileName . . . 1681
70.22.12 TZipFileEntry.UTF8DiskFileName . . . 1681
70.22.13 TZipFileEntry.Size . . . 1681
70.22.14 TZipFileEntry.DateTime . . . 1682
70.22.15 TZipFileEntry.OS . . . 1682
70.22.16 TZipFileEntry.Attributes . . . 1682
70.22.17 TZipFileEntry.CompressionLevel . . . 1682
70.23 TZipper . . . 1682
70.23.1 Description . . . 1682
70.23.2 Method overview . . . 1683
70.23.3 Property overview . . . 1683
70.23.4 TZipper.Create . . . 1683
70.23.5 TZipper.Destroy . . . 1683
70.23.6 TZipper.ZipAllFiles . . . 1684
70.23.7 TZipper.SaveToFile . . . 1684
70.23.8 TZipper.SaveToStream . . . 1684
70.23.9 TZipper.ZipFile . . . 1685
70.23.10 TZipper.ZipFiles . . . 1685
70.23.11 TZipper.Zip . . . 1685
70.23.12 TZipper.Clear . . . 1686
70.23.13 TZipper.Terminate . . . 1686
70.23.14 TZipper.BufferSize . . . 1686
70.23.15 TZipper.OnPercent . . . 1686
70.23.16 TZipper.OnProgress . . . 1687

70.23.17TZipper.OnStartFile . . . 1687
70.23.18TZipper.OnEndFile . . . 1687
70.23.19TZipper.FileName . . . 1687
70.23.20TZipper.FileComment . . . 1687
70.23.21TZipper.Files . . . 1688
70.23.22TZipper.InMemSize . . . 1688
70.23.23TZipper.Entries . . . 1688
70.23.24TZipper.Terminated . . . 1688
70.23.25TZipper.UseLanguageEncoding . . . 1689

71 Reference for unit 'ZStream' 1690
71.1 Used units . . . 1690
71.2 Overview . . . 1690
71.3 Constants, types and variables . . . 1690
71.3.1 Types . . . 1690
71.4 Ecompressionerror . . . 1691
71.4.1 Description . . . 1691
71.5 Edecompressionerror . . . 1691
71.5.1 Description . . . 1691
71.6 Egzfileerror . . . 1691
71.6.1 Description . . . 1691
71.7 Ezliberror . . . 1691
71.7.1 Description . . . 1691
71.8 Tcompressionstream . . . 1691
71.8.1 Description . . . 1691
71.8.2 Method overview . . . 1692
71.8.3 Property overview . . . 1692
71.8.4 Tcompressionstream.create . . . 1692
71.8.5 Tcompressionstream.destroy . . . 1692
71.8.6 Tcompressionstream.write . . . 1692
71.8.7 Tcompressionstream.flush . . . 1693
71.8.8 Tcompressionstream.get_compressionrate . . . 1693
71.8.9 Tcompressionstream.OnProgress . . . 1693
71.9 Tcustomzlibstream . . . 1693
71.9.1 Description . . . 1693
71.9.2 Method overview . . . 1694
71.9.3 Tcustomzlibstream.create . . . 1694
71.9.4 Tcustomzlibstream.destroy . . . 1694
71.10Tdecompressionstream . . . 1694
71.10.1 Description . . . 1694

71.10.2 Method overview 1694
71.10.3 Property overview 1695
71.10.4 Tdecompressionstream.create 1695
71.10.5 Tdecompressionstream.destroy 1695
71.10.6 Tdecompressionstream.read 1695
71.10.7 Tdecompressionstream.Seek 1696
71.10.8 Tdecompressionstream.get_compressionrate 1696
71.10.9 Tdecompressionstream.OnProgress 1696
71.11 TGZFileStream 1697
71.11.1 Description 1697
71.11.2 Method overview 1697
71.11.3 TGZFileStream.create 1697
71.11.4 TGZFileStream.read 1697
71.11.5 TGZFileStream.write 1698
71.11.6 TGZFileStream.seek 1698
71.11.7 TGZFileStream.destroy 1698
71.12 TGZipCompressionStream 1698
71.12.1 Method overview 1698
71.12.2 TGZipCompressionStream.Create 1698
71.12.3 TGZipCompressionStream.Destroy 1699
71.12.4 TGZipCompressionStream.Write 1699
71.13 TGZipDecompressionStream 1699
71.13.1 Method overview 1699
71.13.2 TGZipDecompressionStream.Create 1699
71.13.3 TGZipDecompressionStream.Destroy 1699
71.13.4 TGZipDecompressionStream.Read 1699
71.13.5 TGZipDecompressionStream.Seek 1699

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 FCL (Free Component Library).

Throughout this document, we will refer to functions, types and variables with `typewriter` font. Functions and procedures gave 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 Free Component Library is a series of units that implement various classes and non-visual components for use with Free Pascal. They are building blocks for non-visual and visual programs, such as designed in Lazarus.

The `TDataset` descendents have been implemented in a way that makes them compatible to the Delphi implementation of these units. There are other units that have counterparts in Delphi, but most of them are unique to Free Pascal.

Chapter 1

Reference for unit 'AdvancedIPC'

1.1 Used units

Table 1.1: Used units by unit 'AdvancedIPC'

Name	Page
baseunix	??
Classes	??
singleinstance	1481
System	??
sysutils	??

1.2 Overview

The `AdvancedIPC` unit provides two-way request/response inter-process communication between one server and multiple clients using file-based messaging. The system is non-deterministic - message processing order is not guaranteed. However, `SendRequest` and `PostRequest`/`PeekResponse` sequences from a single client are processed in correct order.

Because it is file based, it should not be used for intensive messaging, but it is suitable for exchanging small messages at startup, for example to transfer control to another instance. It is implemented using files as this is a common denominator on all platforms.

1.3 Constants, types and variables

1.3.1 Resource strings

```
SErrInvalidServerID =
  'Invalid server ID "%s". Please use only alphanumerical characters and underlines.
```

The `SErrInvalidServerID` resource string contains the error message displayed when an invalid server ID is specified. Server IDs must contain only alphanumeric characters and underscores.

```
SErrSetGlobalActive =
  'You cannot change the global property when the server is active.'
```

The SErrSetGlobalActive resource string contains the error message displayed when attempting to change the Global property while the server is active.

```
SErrSetServerIDActive =
  'You cannot change the server ID when the server is active.'
```

The SErrSetServerIDActive resource string contains the error message displayed when attempting to change the ServerID property while the server is active.

1.3.2 Constants

```
HEADER_VERSION = 2
```

The HEADER_VERSION constant defines the version of the message header format. Currently set to 2. Messages with different header versions cannot be exchanged between clients and servers.

1.3.3 Types

```
TFileHandle = Classes.THandle
```

The TFileHandle type is an alias for the platform-specific file handle type from the Classes unit. Used internally for file operations in IPC communication.

```
TMessageType = LongInt
```

The TMessageType is an alias for LongInt used to identify different types of messages in IPC communication. Applications can use different message type values to distinguish between various request and response types.

1.4 TMessageHeader

```
TMessageHeader = packed record
  HeaderVersion : Byte;
  FileLock
  : Byte;
  MsgType : TMessageType;
  MsgLen : Integer;
  MsgVersion
  : Integer;
end
```

The TMessageHeader record defines the structure of the message header used in IPC communication. It contains version information, locking state, message type, length, and version compatibility data.

1.5 EICPException

1.5.1 Description

The EICPException class represents exceptions that occur during IPC operations. It is raised when invalid server IDs are used or when attempting to change properties while the server is active.

1.6 TIPCBase

1.6.1 Description

The `TIPCBase` class provides common functionality for both IPC clients and servers. It handles server identification, global/local communication scope, message versioning, and low-level message operations.

See also: TIPCClient (158), TIPCServer (161)

1.6.2 Method overview

Page	Method	Description
157	FindRunningServers	Find all running servers with matching server ID prefix
157	ServerRunning	Check if a server with given ID is running

1.6.3 Property overview

Page	Properties	Access	Description
158	Global	rw	Communication scope (global or local)
158	MessageVersion	rw	Message version for compatibility checking
158	ServerID	rw	Server identifier for IPC communication

1.6.4 TIPCBase.FindRunningServers

Synopsis: Find all running servers with matching server ID prefix

Declaration:
```
class procedure FindRunningServers(const aServerIDPrefix: string;
                                   const outServerIDs: TStrings;
                                   const aGlobal: Boolean)
```

Visibility: public

Description: The `FindRunningServers` class procedure searches for all currently running IPC servers whose server ID starts with the specified prefix. Results are returned in the `outServerIDs` string list.

See also: ServerRunning (157)

1.6.5 TIPCBase.ServerRunning

Synopsis: Check if a server with given ID is running

Declaration:
```
class function ServerRunning(const aServerID: string;
                             const aGlobal: Boolean) : Boolean;  Overload
```

Visibility: public

Description: The `ServerRunning` class function checks whether an IPC server with the specified ID is currently active. Returns `True` if the server is running and accessible.

See also: FindRunningServers (157)

1.6.6 TIPCBase.ServerID

Synopsis: Server identifier for IPC communication

Declaration: `Property ServerID : string`

Visibility: public

Access: Read,Write

Description: The `ServerID` property specifies the unique identifier for the server. Only alphanumeric characters and underscores are allowed. Clients use this ID to connect to the server.

See also: Global (158)

1.6.7 TIPCBase.Global

Synopsis: Communication scope (global or local)

Declaration: `Property Global : Boolean`

Visibility: public

Access: Read,Write

Description: The `Global` property determines the communication scope. If `True`, processes from different users can communicate. If `False`, only processes from the current user can communicate.

See also: ServerID (158)

1.6.8 TIPCBase.MessageVersion

Synopsis: Message version for compatibility checking

Declaration: `Property MessageVersion : Integer`

Visibility: public

Access: Read,Write

Description: The `MessageVersion` property specifies the message version. Only messages with the same `MessageVersion` can be exchanged between server and client. This ensures compatibility between different versions of applications.

See also: TMessageHeader (156)

1.7 TIPCClient

1.7.1 Description

The `TIPCClient` class provides client-side functionality for inter-process communication. It can send requests to servers and receive responses using either blocking or non-blocking methods.

See also: TIPCBase (157), TIPCServer (161)

1.7.2 Method overview

Page	Method	Description
159	Create	Create IPC client instance
160	DeleteRequest	Delete a request from the queue
160	PeekResponse	Retrieve response from server for a request
159	PostRequest	Post request to server without waiting for response
159	SendRequest	Send request to server and wait until processed
160	ServerRunning	Check if the target server is running

1.7.3 TIPCClient.Create

Synopsis: Create IPC client instance

Declaration:
```
constructor Create(aOwner: TComponent);  Override
```

Visibility: public

Description: The `Create` constructor initializes a new IPC client instance with default settings.

1.7.4 TIPCClient.PostRequest

Synopsis: Post request to server without waiting for response

Declaration:
```
function PostRequest(const aMsgType: TMessageType;
                     const aStream: TStream) : Integer
```

Visibility: public

Description: The `PostRequest` function sends a request to the server and returns immediately without waiting for the server to process it. Returns the request ID that can be used to retrieve the response later using `PeekResponse`.

See also: SendRequest (159), PeekResponse (160)

1.7.5 TIPCClient.SendRequest

Synopsis: Send request to server and wait until processed

Declaration:
```
function SendRequest(const aMsgType: TMessageType;
                     const aStream: TStream; const aTimeOut: Integer)
                      : Boolean
function SendRequest(const aMsgType: TMessageType;
                     const aStream: TStream; const aTimeOut: Integer;
                     out outRequestID: Integer) : Boolean
```

Visibility: public

Description: The `SendRequest` function sends a request to the server and waits until the server has processed it (request file is removed). Returns `True` if the request was processed within the timeout limit.

See also: PostRequest (159), PeekResponse (160)

1.7.6 TIPCClient.PeekResponse

Synopsis: Retrieve response from server for a request

Declaration:
```
function PeekResponse(const aStream: TStream;
                      out outMsgType: TMessageType) : Boolean;  Overload
function PeekResponse(const aStream: TStream;
                      out outMsgType: TMessageType;
                      const aTimeOut: Integer) : Boolean;  Overload
function PeekResponse(const aRequestID: Integer;
                      const aStream: TStream;
                      out outMsgType: TMessageType) : Boolean;  Overload
function PeekResponse(const aRequestID: Integer;
                      const aStream: TStream;
                      out outMsgType: TMessageType;
                      const aTimeOut: Integer) : Boolean;  Overload
```

Visibility: public

Description: The `PeekResponse` function retrieves the server response for a previously sent request. Multiple overloads support immediate checking or waiting with timeout. Can retrieve response for specific request ID or the last request from this client.

See also: PostRequest (159), SendRequest (159)

1.7.7 TIPCClient.DeleteRequest

Synopsis: Delete a request from the queue

Declaration:
```
function DeleteRequest : Boolean;  Overload
function DeleteRequest(const aRequestID: Integer) : Boolean;  Overload
```

Visibility: public

Description: The `DeleteRequest` function removes a request from the server queue. Can delete by request ID or the last request from this client. Returns `True` if the request file existed and was successfully deleted.

See also: PostRequest (159)

1.7.8 TIPCClient.ServerRunning

Synopsis: Check if the target server is running

Declaration:
```
function ServerRunning : Boolean;  Overload
```

Visibility: public

Description: The `ServerRunning` function checks whether the server specified by `ServerID` is currently active and available for communication.

See also: TIPCBase.ServerRunning (157)

1.8 TIPCServer

1.8.1 Description

The `TIPCServer` class provides server-side functionality for inter-process communication. It can receive requests from clients, process them, and send back responses. The server must be started to become available for client connections.

See also: TIPCBase (157), TIPCClient (158)

1.8.2 Method overview

Page	Method	Description
161	Create	Create IPC server instance
164	DeletePendingRequests	Delete all pending requests and responses
162	DeleteRequest	Delete a peeked request
161	Destroy	Destroy IPC server instance
163	FindHighestPendingRequestId	Find the highest pending request ID
163	GetPendingRequestCount	Get the number of pending requests
162	PeekRequest	Peek at incoming requests from clients
163	PostResponse	Send response back to client
162	ReadRequest	Read a previously peeked request
163	StartServer	Start the IPC server
164	StopServer	Stop the IPC server

1.8.3 Property overview

Page	Properties	Access	Description
164	Active	r	Server running state

1.8.4 TIPCServer.Create

Synopsis: Create IPC server instance

Declaration: `constructor Create(aOwner: TComponent);  Override`

Visibility: public

Description: The `Create` constructor initializes a new IPC server instance. The server is not active until `StartServer` is called.

See also: StartServer (163)

1.8.5 TIPCServer.Destroy

Synopsis: Destroy IPC server instance

Declaration: `destructor Destroy;  Override`

Visibility: public

Description: The `Destroy` destructor automatically stops the server if it is active before destroying the instance.

See also: StopServer (164)

1.8.6 TIPCServer.PeekRequest

Synopsis: Peek at incoming requests from clients

Declaration:
```
function PeekRequest(const aStream: TStream;
                     out outMsgType: TMessageType) : Boolean;  Overload
function PeekRequest(const aStream: TStream; out outRequestID: Integer;
                     out outMsgType: TMessageType) : Boolean;  Overload
function PeekRequest(const aStream: TStream; out outRequestID: Integer;
                     out outMsgType: TMessageType;
                     const aTimeOut: Integer) : Boolean;  Overload
function PeekRequest(out outMsgType: TMessageType) : Boolean;  Overload
function PeekRequest(out outRequestID: Integer;
                     out outMsgType: TMessageType) : Boolean;  Overload
function PeekRequest(out outRequestID: Integer;
                     out outMsgType: TMessageType;
                     const aTimeOut: Integer) : Boolean;  Overload
```

Visibility: public

Description: The `PeekRequest` function retrieves incoming requests from clients. Multiple overloads support immediate checking or waiting with timeout. Some overloads read the message content directly, others only peek and require `ReadRequest` to get the content.

See also: ReadRequest (162), PostResponse (163)

1.8.7 TIPCServer.ReadRequest

Synopsis: Read a previously peeked request

Declaration:
```
function ReadRequest(const aRequestID: Integer; const aStream: TStream)
                     : Boolean
```

Visibility: public

Description: The `ReadRequest` function reads the content of a request that was previously peeked using `PeekRequest`. The request must have been peeked but not yet read or deleted.

See also: PeekRequest (162), DeleteRequest (162)

1.8.8 TIPCServer.DeleteRequest

Synopsis: Delete a peeked request

Declaration:
```
function DeleteRequest(const aRequestID: Integer) : Boolean
```

Visibility: public

Description: The `DeleteRequest` function removes a peeked request from the queue without reading it. Returns `True` if the request existed and was successfully deleted.

See also: PeekRequest (162), ReadRequest (162)

1.8.9 TIPCServer.PostResponse

Synopsis: Send response back to client

Declaration:
```
procedure PostResponse(const aRequestID: Integer;
                       const aMsgType: TMessageType;
                       const aStream: TStream)
```

Visibility: public

Description: The `PostResponse` procedure sends a response back to the client for a specific request. The response is identified by the request ID and can be retrieved by the client using `PeekResponse`.

See also: PeekRequest (162), TIPCClient.PeekResponse (160)

1.8.10 TIPCServer.FindHighestPendingRequestId

Synopsis: Find the highest pending request ID

Declaration:
```
function FindHighestPendingRequestId : Integer
```

Visibility: public

Description: The `FindHighestPendingRequestId` function returns the highest request ID among all pending requests in the server queue. Returns -1 if no pending requests exist.

See also: GetPendingRequestCount (163)

1.8.11 TIPCServer.GetPendingRequestCount

Synopsis: Get the number of pending requests

Declaration:
```
function GetPendingRequestCount : Integer
```

Visibility: public

Description: The `GetPendingRequestCount` function returns the total number of requests currently waiting in the server queue.

See also: FindHighestPendingRequestId (163)

1.8.12 TIPCServer.StartServer

Synopsis: Start the IPC server

Declaration:
```
function StartServer(const aDeletePendingRequests: Boolean) : Boolean
```

Visibility: public

Description: The `StartServer` function activates the IPC server, making it available for client connections. Returns `True` if the server was successfully started or was already running. Optionally deletes any pending requests from previous sessions.

See also: StopServer (164), Active (164)

1.8.13 TIPCServer.StopServer

Synopsis: Stop the IPC server

Declaration: `function StopServer(const aDeletePendingRequests: Boolean) : Boolean`

Visibility: public

Description: The `StopServer` function deactivates the IPC server, making it unavailable for client connections. Returns `True` if the server was successfully stopped. Optionally deletes any remaining pending requests.

See also: StartServer (163), Active (164)

1.8.14 TIPCServer.DeletePendingRequests

Synopsis: Delete all pending requests and responses

Declaration: `procedure DeletePendingRequests`

Visibility: public

Description: The `DeletePendingRequests` procedure removes all pending requests and responses from the server queue. This is useful for cleanup operations during server startup or shutdown.

See also: StartServer (163), StopServer (164)

1.8.15 TIPCServer.Active

Synopsis: Server running state

Declaration: `Property Active : Boolean`

Visibility: public

Access: Read

Description: The `Active` property indicates whether the IPC server is currently running and available for client connections. `True` if the server has been started, `False` otherwise.

See also: StartServer (163), StopServer (164)

1.9 TReleaseHandleStream

1.9.1 Description

The `TReleaseHandleStream` class extends `THandleStream` to automatically close the file handle when the stream is destroyed.

1.9.2 Method overview

Page	Method	Description
165	Destroy	Destructor that closes the file handle

1.9.3 TReleaseHandleStream.Destroy

Synopsis: Destructor that closes the file handle

Declaration: `destructor Destroy; Override`

Visibility: public

Description: The `Destroy` destructor closes the file handle before calling the inherited destructor.

Chapter 2

Reference for unit 'AdvancedSingleInstance'

2.1 Used units

Table 2.1: Used units by unit 'AdvancedSingleInstance'

Name	Page
AdvancedIPC	155
Classes	??
singleinstance	1481
System	??
sysutils	??

2.2 Overview

The `AdvancedSingleInstance` unit contains an implementation of the TBaseSingleInstance (1482) class, based on functionality found in the AdvcancedIPC (**??**) unit. The class is called TAdvancedSingleInstance (167)

2.3 Constants, types and variables

2.3.1 Types

```
TSingleInstanceReceivedCustomMessage = procedure
  (Sender: TBaseSingleInstance;
  MsgID: Integer;
  MsgType: Integer
  ;
                                                MsgData: TStream
  )
                                                 of object
```

TSingleInstanceReceivedCustomMessage is the signature of the TAdvancedSingleInstance.OnServerReceivedCustomRequest (170) event. It carries the information received when the server receives a custom message.

2.4 TAdvancedSingleInstance

2.4.1 Description

TAdvancedSingleInstance is a TBaseSingleInstance (166) descendent. It implements the required functionality using the IPC mechanism of the advancedipc (166) unit: it uses files to communicate. This ensures the mechanism works on all platforms.

See also: TBaseSingleInstance (166), advancedipc (166)

2.4.2 Method overview

Page	Method	Description
170	ClientPeekCustomResponse	Check whether the server posted a response.
169	ClientPostCustomRequest	Send a custom message to the server.
168	ClientPostParams	Post client command-line parameters to the server.
169	ClientSendCustomRequest	ClientSendCustomRequest.
167	Create	Create a new instance of TAdvancedSingleInstance.
168	ServerCheckMessages	Check for incoming messages for the server.
169	ServerPostCustomResponse	Post a server response to the client.
168	Start	Start IPC communication and check the type of the application instance.
168	Stop	End IPC communication.

2.4.3 Property overview

Page	Properties	Access	Description
170	Global	rw	Is the instance global (for all users) or local (for the current user).
170	ID	rw	The ID of this instance.
170	OnServerReceivedCustomRequest	rw	Server event truggered when a custom request is received.

2.4.4 TAdvancedSingleInstance.Create

Synopsis: Create a new instance of TAdvancedSingleInstance.

Declaration: `constructor Create(aOwner: TComponent); Override`

Visibility: public

Description: Create calls the inherited constructor, and then constructs the ID (166) property which identifies the application.

See also: ID (166)

2.4.5 TAdvancedSingleInstance.Start

Synopsis: Start IPC communication and check the type of the application instance.

Declaration: `function Start : TSingleInstanceStart; Override`

Visibility: public

Description: `Start` implements the abstract TBaseSingleInstance.Start (166) call. It initiates the IPC mechanism.

See also: TBaseSingleInstance.Start (166), TAdvancedSingleInstance.Stop (168)

2.4.6 TAdvancedSingleInstance.Stop

Synopsis: End IPC communication.

Declaration: `procedure Stop; Override`

Visibility: public

Description: `Start` implements the abstract TBaseSingleInstance.Stop (166) call. It terminates the IPC mechanism.

See also: TBaseSingleInstance.Stop (166), TAdvancedSingleInstance.Start (168)

2.4.7 TAdvancedSingleInstance.ServerCheckMessages

Synopsis: Check for incoming messages for the server.

Declaration: `procedure ServerCheckMessages; Override`

Visibility: public

Description: `ServerCheckMessages` implements TAdvancedSingleInstance.ServerCheckMessages (168) and checks if there are any messages in the message queue using the API of advancedIPC (166). If there are messages, it fires the OnServerReceivedParams (1481) event.

See also: TAdvancedSingleInstance.ServerCheckMessages (168), TAdvancedSingleInstance.Start (168), TAdvancedSingleInstance.Stop (168), OnServerReceivedParams (1481)

2.4.8 TAdvancedSingleInstance.ClientPostParams

Synopsis: Post client command-line parameters to the server.

Declaration: `procedure ClientPostParams; Override`

Visibility: public

Description: `ClientPostParams` implements TAdvancedSingleInstance.ClientPostParams (168). It sends the command-line parameters using ClientPostCustomRequest (169).

Errors: If this application instance is not a client instance, then a ESingleInstance (166) exception is raised.

See also: TAdvancedSingleInstance.ClientPostParams (168), ClientPostCustomRequest (169)

2.4.9 TAdvancedSingleInstance.ClientPostCustomRequest

Synopsis: Send a custom message to the server.

Declaration:
```
function ClientPostCustomRequest(const aMsgType: Integer;
                                 const aStream: TStream) : Integer
```

Visibility: public

Description: ClientPostCustomRequest> sends the data in aStream to the server instance using message type aMsgType. It returns the message id.

Errors: If this application instance is not a client instance, then a ESingleInstance (166) exception is raised.

See also: TAdvancedSingleInstance.ClientSendCustomRequest (169), TAdvancedSingleInstance.ServerPostCustomResponse (169)

2.4.10 TAdvancedSingleInstance.ClientSendCustomRequest

Synopsis: ClientSendCustomRequest.

Declaration:
```
function ClientSendCustomRequest(const aMsgType: Integer;
                                 const aStream: TStream) : Boolean
                                 ;  Overload
function ClientSendCustomRequest(const aMsgType: Integer;
                                 const aStream: TStream;
                                 out outRequestID: Integer) : Boolean
                                 ;  Overload
```

Visibility: public

Description: ClientPostCustomRequest> sends the data in aStream to the server instance using message type aMsgType. It returns True if the message was sent successfully. It returns the message id in outRequestID, if supplied.

Errors: If this application instance is not a client instance, then a ESingleInstance (166) exception is raised.

See also: TAdvancedSingleInstance.ClientPostCustomRequest (169), TAdvancedSingleInstance.ServerPostCustomResponse (169)

2.4.11 TAdvancedSingleInstance.ServerPostCustomResponse

Synopsis: Post a server response to the client.

Declaration:
```
procedure ServerPostCustomResponse(const aRequestID: Integer;
                                   const aMsgType: Integer;
                                   const aStream: TStream)
```

Visibility: public

Description: ClientPostCustomRequest> sends the data in aStream to the client instance using message type aMsgType, in response to client message aRequestID. The client can check for the response using TAdvancedSingleInstance.ClientPeekCustomResponse (170)

Errors: If this application instance is not a server instance, then a ESingleInstance (166) exception is raised.

See also: TAdvancedSingleInstance.ClientPostCustomRequest (169), TAdvancedSingleInstance.ClientSendCustomRequest (169), TAdvancedSingleInstance.ClientPeekCustomResponse (170)

2.4.12 TAdvancedSingleInstance.ClientPeekCustomResponse

Synopsis: Check whether the server posted a response.

Declaration:
```
function ClientPeekCustomResponse(const aStream: TStream;
                                  out outMsgType: Integer) : Boolean
```

Visibility: public

Description: `ClientPeekCustomResponse` checks if the server posted any responses. It returns `True` if there was a message, and returns the message type in `outMesgType`, the payload is read into the stream `aStream`. The message timeout TimeoutMessages (166) is observed: if no message arrived before the timeout expires the call will return `False`.

Errors: If this application instance is not a client instance, then a ESingleInstance (166) exception is raised. The `aStream` must be non-nil, or an access violation will occur.

See also: TimeoutMessages (166), TAdvancedSingleInstance.ServerPostCustomResponse (169)

2.4.13 TAdvancedSingleInstance.ID

Synopsis: The ID of this instance.

Declaration:
```
Property ID : AnsiString
```

Visibility: public

Access: Read,Write

Description: `ID` is the unique ID for this application instance.

See also: TBaseSingleInstance.ID (166)

2.4.14 TAdvancedSingleInstance.Global

Synopsis: Is the instance global (for all users) or local (for the current user).

Declaration:
```
Property Global : Boolean
```

Visibility: public

Access: Read,Write

Description: `Global` indicates whether the instance is global (for all users) or local (for the current user only). This must be set before Start (168) is called.

See also: Start (168)

2.4.15 TAdvancedSingleInstance.OnServerReceivedCustomRequest

Synopsis: Server event truggered when a custom request is received.

Declaration:
```
Property OnServerReceivedCustomRequest : TSingleInstanceReceivedCustomMessage
```

Visibility: public

Access: Read,Write

Description: `OnServerReceivedCustomRequest` is triggered when a custom request (i.e. not a command-line parameters request) is received by the server.

See also: TSingleInstanceReceivedCustomMessage (166)

Chapter 3

Reference for unit 'ascii85'

3.1 Used units

Table 3.1: Used units by unit 'ascii85'

Name	Page
Classes	??
System	??
sysutils	??

3.2 Overview

The ascii85 provides an ASCII 85 or base 85 decoding algorithm. It is class and stream based: the TASCII85DecoderStream (172) stream can be used to decode any stream with ASCII85 encoded data.

Currently, no ASCII85 encoder stream is available.

It's usage and purpose is similar to the IDEA (1258) or base64 (200) units.

3.3 Constants, types and variables

3.3.1 Types

```
TASCII85State = (ascInitial,ascOneEncodedChar,ascTwoEncodedChars,
  ascThreeEncodedChars,ascFourEncodedChars,
  ascNoEncodedChar,ascPrefix)
```

Table 3.2: Enumeration values for type TASCII85State

Value	Explanation
ascFourEncodedChars	Four encoded characters in buffer.
ascInitial	Initial state.
ascNoEncodedChar	No encoded characters in buffer.
ascOneEncodedChar	One encoded character in buffer.
ascPrefix	Prefix processing.
ascThreeEncodedChars	Three encoded characters in buffer.
ascTwoEncodedChars	Two encoded characters in buffer.

`TASCII85State` is for internal use, it contains the current state of the decoder.

3.4 TASCII85DecoderStream

3.4.1 Description

`TASCII85DecoderStream` is a read-only stream: it takes an input stream with ASCII 85 encoded data, and decodes the data as it is read. To this end, it overrides the TSTream.Read (**??**) method.

The stream cannot be written to, trying to write to the stream will result in an exception.

3.4.2 Method overview

Page	Method	Description
173	Close	Close decoder.
173	ClosedP	Check if the state is correct.
172	Create	Create new ASCII 85 decoder stream.
173	Decode	Decode source byte.
173	Destroy	Clean up instance.
174	Read	Read data from stream.
174	Seek	Set stream position.

3.4.3 Property overview

Page	Properties	Access	Description
174	BExpectBoundary	rw	Expect c̃haracter.

3.4.4 TASCII85DecoderStream.Create

Synopsis: Create new ASCII 85 decoder stream.

Declaration: `constructor Create(aStream: TStream)`

Visibility: published

Description: `Create` instantiates a new `TASCII85DecoderStream` instance, and sets `aStream` as the source stream.

See also: TASCII85DecoderStream.Destroy (173)

3.4.5 TASCII85DecoderStream.Decode

Synopsis: Decode source byte.

Declaration: `procedure Decode(aInput: Byte)`

Visibility: published

Description: `Decode` decodes a source byte, and transfers it to the buffer. It is an internal routine and should not be used directly.

See also: TASCII85DecoderStream.Close (173)

3.4.6 TASCII85DecoderStream.Close

Synopsis: Close decoder.

Declaration: `procedure Close`

Visibility: published

Description: `Close` closes the decoder mechanism: it checks if all data was read and performs a check to see whether all input data was consumed.

Errors: If the input stream was invalid, an `EConvertError` exception is raised.

See also: TASCII85DecoderStream.ClosedP (173), TASCII85DecoderStream.Read (174), TASCII85DecoderStream.Destroy (173)

3.4.7 TASCII85DecoderStream.ClosedP

Synopsis: Check if the state is correct.

Declaration: `function ClosedP : Boolean`

Visibility: published

Description: `ClosedP` checks if the decoder state is one of `ascInitial`, `ascNoEncodedChar`, `ascPrefix`, and returns `True` if it is.

See also: TASCII85DecoderStream.Close (173), TASCII85DecoderStream.BExpectBoundary (174)

3.4.8 TASCII85DecoderStream.Destroy

Synopsis: Clean up instance.

Declaration: `destructor Destroy;  Override`

Visibility: public

Description: `Destroy` closes the input stream using Close (173) and cleans up the `TASCII85DecoderStream` instance from memory.

Errors: In case the input stream was invalid, an exception may occur.

See also: TASCII85DecoderStream.Close (173)

3.4.9 TASCII85DecoderStream.Read

Synopsis: Read data from stream.

Declaration: `function Read(var aBuffer; aCount: LongInt) : LongInt;  Override`

Visibility: public

Description: `Read` attempts to read `aCount` bytes from the stream and places them in `aBuffer`. It reads only as much data as is available. The actual number of read bytes is returned.

The read method reads as much data from the input stream as needed to get to `aCount` bytes, in general this will be `aCount*5/4` bytes.

3.4.10 TASCII85DecoderStream.Seek

Synopsis: Set stream position.

Declaration:
```
function Seek(aOffset: LongInt; aOrigin: Word) : LongInt;  Override
function Seek(const aOffset: Int64; aOrigin: TSeekOrigin) : Int64
             ;  Override;  Overload
```

Visibility: public

Description: `Seek` sets the stream position. It only allows to set the position to the current position of this file, and returns then the current position. All other arguments will result in an `EReadError` exception.

Errors: In case the arguments are different from `soCurrent` and 0, an `EReadError` exception will be raised.

See also: TASCII85DecoderStream.Read (174)

3.4.11 TASCII85DecoderStream.BExpectBoundary

Synopsis: Expect c̃haracter.

Declaration: `Property BExpectBoundary : Boolean`

Visibility: published

Access: Read,Write

Description: `BExpectBoundary` is `True` if a encoded data boundary is to be expected ("≿").

See also: ClosedP (173)

3.5 TASCII85EncoderStream

3.5.1 Description

`TASCII85EncoderStream` is the counterpart to the TASCII85DecoderStream (172) decoder stream: what `TASCII85EncoderStream` encodes, can be decoded by TASCII85DecoderStream (172).

The encoder stream works using a destination stream: whatever data is written to the encoder stream is encoded and written to the destination stream. The stream must be passed on in the constructor.

Note that all encoded data is only written to the destination stream when the encoder stream is destroyed.

See also: TASCII85EncoderStream.create (175), TASCII85DecoderStream (172)

3.5.2 Method overview

Page	Method	Description
175	Create	Create a new instance of `TASCII85EncoderStream`.
175	Destroy	Flushed the data to the output stream and cleans up the encoder instance.
175	Write	Write data encoded to the destination stream.

3.5.3 Property overview

Page	Properties	Access	Description
176	Boundary	r	Is a boundary delineator written before and after the data.
176	Width	r	Width of the lines written to the data stream.

3.5.4 TASCII85EncoderStream.Create

Synopsis: Create a new instance of `TASCII85EncoderStream`.

Declaration: `constructor Create(ADest: TStream; AWidth: Integer; ABoundary: Boolean)`

Visibility: public

Description: `Create` creates a new instance of`TASCII85EncoderStream`. It stores `ADest` as the destination stream for the encoded data. The `Width` parameter indicates the width of the lines that are written by the encoder: after this amount of characters, a linefeed is put in the data stream. If `ABoundary` is `True` then a boundary delineator is written to the stream before and after the data.

See also: TASCII85EncoderStream (174), Width (176), Boundary (176)

3.5.5 TASCII85EncoderStream.Destroy

Synopsis: Flushed the data to the output stream and cleans up the encoder instance.

Declaration: `destructor Destroy;  Override`

Visibility: public

Description: `Destroy` writes the data remaining in the internal buffer to the destination stream (possibly followed by a boundary delineator) and then destroys the encoder instance.

See also: TASCII85EncoderStream.Write (175), TASCII85EncoderStream.Boundary (176)

3.5.6 TASCII85EncoderStream.Write

Synopsis: Write data encoded to the destination stream.

Declaration: `function Write(const aBuffer; aCount: LongInt) : LongInt;  Override`

Visibility: public

Description: `Write` encodes the `aCount` bytes of data in `aBuffer` and writes the encoded data to the destination stream.

Not all data is written immediately to the destination stream. Only after the encoding stream is destroyed will the destination stream contain the full data.

See also: TASCII85EncoderStream.Destroy (175)

3.5.7 TASCII85EncoderStream.Width

Synopsis: Width of the lines written to the data stream.

Declaration: `Property Width : Integer`

Visibility: public

Access: Read

Description: `Width` is the width of the lines of encoded data written to the stream. After `Width` lines, a line ending will be written to the stream. The value is passed to the constructor and cannot be changed afterwards.

See also: Boundary (176), Create (175)

3.5.8 TASCII85EncoderStream.Boundary

Synopsis: Is a boundary delineator written before and after the data.

Declaration: `Property Boundary : Boolean`

Visibility: public

Access: Read

Description: `Boundary` indicates whether the stream will write a boundary delineator before and after the encoded data. It is passed to the constructor and cannot be changed.

See also: Width (176), Create (175)

3.6 TASCII85RingBuffer

3.6.1 Description

`TASCII85RingBuffer` is an internal buffer class: it maintains a memory buffer of 1Kb, for faster reading of the stream. It should not be necessary to instantiate an instance of this class, the TASCII85DecoderStream (172) decoder stream will create an instance of this class automatically.

See also: TASCII85DecoderStream (172)

3.6.2 Method overview

Page	Method	Description
177	Read	Read data from the internal buffer.
177	Write	Write data to the internal buffer.

3.6.3 Property overview

Page	Properties	Access	Description
177	FillCount	r	Number of bytes in buffer.
177	Size	r	Size of buffer.

3.6.4 TASCII85RingBuffer.Write

Synopsis: Write data to the internal buffer.

Declaration: `procedure Write(const aBuffer; aSize: Cardinal)`

Visibility: published

Description: `Write` writes `aSize` bytes from `aBuffer` to the internal memory buffer. Only as much bytes are written as will fit in the buffer.

See also: TASCII85RingBuffer.FillCount (177), TASCII85RingBuffer.Read (177), TASCII85RingBuffer.Size (177)

3.6.5 TASCII85RingBuffer.Read

Synopsis: Read data from the internal buffer.

Declaration: `function Read(var aBuffer; aSize: Cardinal) : Cardinal`

Visibility: published

Description: `Read` will read `aSize` bytes from the internal buffer and writes them to `aBuffer`. If not enough bytes are available, only as much bytes as available will be written. The function returns the number of bytes transferred.

See also: TASCII85RingBuffer.FillCount (177), TASCII85RingBuffer.Write (177), TASCII85RingBuffer.Size (177)

3.6.6 TASCII85RingBuffer.FillCount

Synopsis: Number of bytes in buffer.

Declaration: `Property FillCount : Cardinal`

Visibility: published

Access: Read

Description: `FillCount` is the available amount of bytes in the buffer.

See also: TASCII85RingBuffer.Write (177), TASCII85RingBuffer.Read (177), TASCII85RingBuffer.Size (177)

3.6.7 TASCII85RingBuffer.Size

Synopsis: Size of buffer.

Declaration: `Property Size : Cardinal`

Visibility: published

Access: Read

Description: `Size` is the total size of the memory buffer. This is currently hardcoded to 1024Kb.

See also: TASCII85RingBuffer.FillCount (177)

Chapter 4

Reference for unit 'AVL_Tree'

4.1 Used units

Table 4.1: Used units by unit 'AVL_Tree'

Name	Page
Classes	??
System	??
sysutils	??

4.2 Overview

The `avl_tree` unit implements a general-purpose AVL (balanced) tree class: the TAVLTree (179) class and it's associated data node class TAVLTreeNode (193).

4.3 Constants, types and variables

4.3.1 Types

```
PAVLTreeNode = ^TAVLTreeNode
```

Pointer to TAVLTreeNode.

```
TAVLTreeClass = Class of TAVLTree
```

`TAVLTreeClass` is the class of TAVLTree (179).

```
TAVLTreeNodeClass = Class of TAVLTreeNode
```

`TAVLTreeNodeClass` is the class of TAVLTreeNode (193). It is the type of the TAVLTree.NodeClass (192) property and determines what class of nodes will be created by the tree.

```
TObjectSortCompare = function(Tree: TAVLTree; Data1: Pointer;
  Data2: Pointer) : Integer of object
```

TObjectSortCompare is the prototype for the TAVLTree.OnObjectCompare (192) property. When assigned, it is used to sort the elements in the tree. It provides more information than the standard TListSortCompare handler used in TAVLTree.OnCompare (192): it also passes the tree to the sort mechanism.

4.3.2 Variables

NodeMemManager : TAVLTreeNodeMemManager

NodeMemManager is the default node manager for a new instance of TAVLTree.

4.4 TAVLTree

4.4.1 Description

TAVLTree maintains a balanced AVL tree. The tree consists of TAVLTreeNode (193) nodes, each of which has a Data pointer associated with it. The TAVLTree component offers methods to balance and search the tree.

By default, the list is searched with a simple pointer comparison algorithm, but a custom search mechanism can be specified in the OnCompare (192) property.

See also: TAVLTreeNode (193)

4.4.2 Method overview

Page	Method	Description
182	Add	Add a new node to the tree.
183	AddAscendingSequence	
186	Assign	Assign another tree.
184	Clear	Clears the tree.
186	Compare	Compare 2 nodes.
191	ConsistencyCheck	Check the consistency of the tree.
181	Create	Create a new instance of `TAVLTree`.
181	CreateObjectCompare	Create an instance of the tree with extended compare method.
183	Delete	Delete a node from the tree.
181	Destroy	Destroy the `TAVLTree` instance.
182	DisposeNode	Dispose of a node outside of the tree.
185	Equals	Check if two trees are equal.
186	Find	Find a data item in the tree.
188	FindHighest	Find the highest (rightmost) node in the tree.
186	FindKey	Find a data item in the tree using alternate compare mechanism.
189	FindLeftMost	Find the node most left to a specified data node.
189	FindLeftMostKey	Find the node most left to a specified key node.
190	FindLeftMostSameKey	Find the node most left to a specified node with the same data.
188	FindLowest	Find the lowest (leftmost) node in the tree.
188	FindNearest	Find the node closest to the data in the tree.
187	FindNearestKey	Find nearest key for a data pointer.
188	FindPointer	Search for a data pointer.
187	FindPrecessor	
189	FindRightMost	Find the node most right to a specified node.
189	FindRightMostKey	Find the node most right to a specified key node.
190	FindRightMostSameKey	Find the node most right of a specified node with the same data.
187	FindSuccessor	Find successor to node.
185	FreeAndClear	Clears the tree and frees nodes.
185	FreeAndDelete	Delete a node from the tree and destroy it.
190	GetEnumerator	Get an enumerator for the tree.
190	GetEnumeratorHighToLow	Return an enumerator that enumerates the tree in reversed order.
185	IsEqual	Check whether 2 tree instances are equal.
184	MoveDataLeftMost	Move data to the nearest left element.
184	MoveDataRightMost	Move data to the nearest right element.
182	NewNode	Create a new tree node.
191	NodeToReportStr	Create a textual dump of the tree.
183	Remove	Remove a data item from the list.
184	RemovePointer	Remove a pointer item from the list.
191	ReportAsString	Return the tree report as a string.
181	SetNodeManager	Set the node instance manager to use.
191	WriteReportToStream	Write the contents of the tree consistency check to the stream.

4.4.3 Property overview

Page	Properties	Access	Description
193	Count	r	Number of nodes in the tree.
192	NodeClass	rw	Node class to create.
192	OnCompare	rw	Compare function used when comparing nodes.
192	OnObjectCompare	rw	Compare handler.
192	Root	r	Root node of the tree.

4.4.4 TAVLTree.Create

Synopsis: Create a new instance of `TAVLTree`.

Declaration:
```
constructor Create(const OnCompareMethod: TListSortCompare)
constructor Create
```

Visibility: public

Description: `Create` initializes a new instance of TAVLTree (179). An alternate OnCompare (192) can be provided: the default `OnCompare` method compares the 2 data pointers of a node.

See also: OnCompare (192)

4.4.5 TAVLTree.CreateObjectCompare

Synopsis: Create an instance of the tree with extended compare method.

Declaration:
```
constructor CreateObjectCompare
                                    (const OnCompareMethod: TObjectSortCompare)
```

Visibility: public

Description: `CreateObjectCompare` is an alternative constructor that accepts a TObjectSortCompare (179) compare function instead of a regular `TListSortCompare` compare function. The compare function can still be set in the TAVLTree.OnObjectCompare (192) property.

See also: TAVLTree.OnObjectCompare (192)

4.4.6 TAVLTree.Destroy

Synopsis: Destroy the `TAVLTree` instance.

Declaration:
```
destructor Destroy;  Override
```

Visibility: public

Description: `Destroy` clears the nodes (the node data is not freed) and then destroys the `TAVLTree` instance.

See also: TAVLTree.Clear (184), TAVLTree.Create (181)

4.4.7 TAVLTree.SetNodeManager

Synopsis: Set the node instance manager to use.

Declaration:
```
procedure SetNodeManager(NewMgr: TBaseAVLTreeNodeManager;
                         AutoFree: Boolean)
```

Visibility: public

Description: `SetNodeManager` sets the node manager instance used by the tree to `newmgr`. It should be called before any nodes are added to the tree. The `TAVLTree` instance will not destroy the nodemanager, thus the same instance of the tree node manager can be used to manager the nodes of multiple `TAVLTree` instances.

By default, a single instance of TAVLTreeNodeMemManager (196) is used to manage the nodes of all `TAVLTree` instances.

See also: TBaseAVLTreeNodeManager (198), TAVLTreeNodeMemManager (196)

4.4.8 TAVLTree.NewNode

Synopsis: Create a new tree node.

Declaration:
```
function NewNode : TAVLTreeNode;  Virtual
```

Visibility: public

Description: `NewNode` creates a new node, but does not insert it in the tree. It will use the node manager if that is set. If it is not set then the TAVLTree.NodeClass (192) class is used to create a new node.

See also: TAVLTree.NodeClass (192), TAVLTree.Add (182), TAVLTree.DisposeNode (182)

4.4.9 TAVLTree.DisposeNode

Synopsis: Dispose of a node outside of the tree.

Declaration:
```
procedure DisposeNode(ANode: TAVLTreeNode);  Virtual
```

Visibility: public

Description: `DisposeNode` disposes of a node outside of the tree. If the node manager is set, the node is returned to the manager, otherwise it is freed. Do not use this on a node that is still in the tree.

Errors: If use on a node in the tree, no error will happen, but the tree will no longer be correct and access violations may happen later on.

See also: TAVLTree.NewNode (182)

4.4.10 TAVLTree.Add

Synopsis: Add a new node to the tree.

Declaration:
```
procedure Add(ANode: TAVLTreeNode)
function Add(Data: Pointer) : TAVLTreeNode
```

Visibility: public

Description: `Add` adds a new `Data` or `Node` to the tree. It inserts the node so that the tree is maximally balanced by rebalancing the tree after the insert. In case a `data` pointer is added to the tree, then the node that was created is returned.

See also: TAVLTree.Delete (183), TAVLTree.Remove (183)

4.4.11 TAVLTree.AddAscendingSequence

Synopsis:

Declaration:
```
function AddAscendingSequence(Data: Pointer; LastAdded: TAVLTreeNode;
                              var Successor: TAVLTreeNode) : TAVLTreeNode
```

Visibility: public

Description: `AddAscendingSequence` is an optimized version of Add (182) for quickly adding an ascending sequence of nodes. It adds `Data` between `LastAdded` and `Successor` as a state and skips searching for an insert position. For nodes with same value the order of the sequence is kept.

It can be used as follows:

```
LastNode:=nil; // TAvlTreeNode
Successor:=nil; // TAvlTreeNode
for i:=1 to 1000 do
 LastNode:=Tree.AddAscendingSequence(TItem.Create(i),LastNode,Successor);
```

If `LastAdded` is `Nil` a regular add is performed.

Errors: If the nodes are not in ascending order, the tree will not be consistent.

See also: TAVLTree.Add (182)

4.4.12 TAVLTree.Delete

Synopsis: Delete a node from the tree.

Declaration:
```
procedure Delete(ANode: TAVLTreeNode)
```

Visibility: public

Description: `Delete` removes the node from the tree. The node is not freed, but is passed to a TAVLTreeNode-MemManager (196) instance for future reuse. The data that the node represents is also not freed.

The tree is rebalanced after the node was deleted.

See also: TAVLTree.Remove (183), TAVLTree.RemovePointer (184), TAVLTree.Clear (184)

4.4.13 TAVLTree.Remove

Synopsis: Remove a data item from the list.

Declaration:
```
function Remove(Data: Pointer) : Boolean
```

Visibility: public

Description: `Remove` finds the node associated with `Data` using find (186) and, if found, deletes it from the tree. Only the first occurrence of `Data` will be removed.

See also: TAVLTree.Delete (183), TAVLTree.RemovePointer (184), TAVLTree.Clear (184), TAVLTree.Find (186)

4.4.14 TAVLTree.RemovePointer

Synopsis: Remove a pointer item from the list.

Declaration: `function RemovePointer(Data: Pointer) : Boolean`

Visibility: public

Description: `Remove` uses FindPointer (188) to find the node associated with the pointer `Data` and, if found, deletes it from the tree. Only the first occurrence of `Data` will be removed.

See also: TAVLTree.Remove (183), TAVLTree.Delete (183), TAVLTree.Clear (184)

4.4.15 TAVLTree.MoveDataLeftMost

Synopsis: Move data to the nearest left element.

Declaration: `procedure MoveDataLeftMost(var ANode: TAVLTreeNode)`

Visibility: public

Description: `MoveDataLeftMost` moves the data from the node `ANode` to the nearest left location relative to `Anode`. It returns the new node where the data is positioned. The data from the former left node will be switched to `ANode`.

This operation corresponds to switching the current with the previous element in a list.

See also: TAVLTree.MoveDataRightMost (184)

4.4.16 TAVLTree.MoveDataRightMost

Synopsis: Move data to the nearest right element.

Declaration: `procedure MoveDataRightMost(var ANode: TAVLTreeNode)`

Visibility: public

Description: `MoveDataRightMost` moves the data from the node `ANode` to the rightmost location relative to `Anode`. It returns the new node where the data is positioned. The data from the former rightmost node will be switched to `ANode`.

This operation corresponds to switching the current with the next element in a list.

See also: TAVLTree.MoveDataLeftMost (184)

4.4.17 TAVLTree.Clear

Synopsis: Clears the tree.

Declaration: `procedure Clear`

Visibility: public

Description: `Clear` deletes all nodes from the tree. The nodes themselves are not freed, and the data pointer in the nodes is also not freed.

If the node's data must be freed as well, use TAVLTree.FreeAndClear (185) instead.

See also: TAVLTree.FreeAndClear (185), TAVLTree.Delete (183)

4.4.18 TAVLTree.FreeAndClear

Synopsis: Clears the tree and frees nodes.

Declaration: `procedure FreeAndClear`

Visibility: public

Description: `FreeAndClear` deletes all nodes from the tree. The data pointer in the nodes is assumed to be an object, and is freed prior to deleting the node from the tree.

See also: TAVLTree.Clear (184), TAVLTree.Delete (183), TAVLTree.FreeAndDelete (185)

4.4.19 TAVLTree.FreeAndDelete

Synopsis: Delete a node from the tree and destroy it.

Declaration: `procedure FreeAndDelete(ANode: TAVLTreeNode);  Virtual`

Visibility: public

Description: `FreeAndDelete` deletes a node from the tree, and destroys the data pointer: The data pointer in the nodes is assumed to be an object, and is freed by calling its destructor.

See also: TAVLTree.Clear (184), TAVLTree.Delete (183), TAVLTree.FreeAndClear (185)

4.4.20 TAVLTree.Equals

Synopsis: Check if two trees are equal.

Declaration: `function Equals(Obj: TObject) : Boolean;  Override`

Visibility: public

Description: `Equals` checks, when passed an TAVLTtree, whether the tree is equal (using TAvlTree.IsEqual (185), comparing keys only). If another type of object is passed, the inherited `IsEqual` is called.

Errors: None.

See also: TAvlTree.IsEqual (185)

4.4.21 TAVLTree.IsEqual

Synopsis: Check whether 2 tree instances are equal.

Declaration: `function IsEqual(aTree: TAVLTree; CheckDataPointer: Boolean) : Boolean`

Visibility: public

Description: `IsEqual` checks the current tree with `aTree` and checks whether the two trees contain the same data in the same order and whether they use the same compare methods, and node class. If `CheckDataPointer` is `True`, only the data pointers are compared, not the keys.

Errors: None.

See also: TAvlTree.Equals (185)

4.4.22 TAVLTree.Assign

Synopsis: Assign another tree.

Declaration:
```
procedure Assign(aTree: TAVLTree);  Virtual
```

Visibility: public

Description: `Assign` copies all data from `aTree` to the current tree if they are not equal. The current tree is cleared first. Note that the compare function(s) and class node are not copied, only the data.

Errors: If you pass nil, an exception is raised.

See also: TAVLTree.IsEqual (185)

4.4.23 TAVLTree.Compare

Synopsis: Compare 2 nodes.

Declaration:
```
function Compare(Data1: Pointer; Data2: Pointer) : Integer
```

Visibility: public

Description: `Compare` compares the keys from 2 data pointers. It uses the appropriate compare function TAVLtree.OnCompare (192) or TAVLTree.OnObjectCompare (192) to do so. The result is

- negative if the first key comes before the second
- 0 when the two keys are equal.
- positive if the second key comes before the first

See also: TAVLTree.OnObjectCompare (192), TAVLtree.OnCompare (192)

4.4.24 TAVLTree.Find

Synopsis: Find a data item in the tree.

Declaration:
```
function Find(Data: Pointer) : TAVLTreeNode
```

Visibility: public

Description: `Find` uses the default OnCompare (192) comparing function to find the `Data` pointer in the tree. It returns the `TAVLTreeNode` instance that results in a successful compare with the `Data` pointer, or `Nil` if none is found.

The default `OnCompare` function compares the actual pointers, which means that by default `Find` will give the same result as FindPointer (188).

See also: OnCompare (192), FindKey (186)

4.4.25 TAVLTree.FindKey

Synopsis: Find a data item in the tree using alternate compare mechanism.

Declaration:
```
function FindKey(Key: Pointer;
                 const OnCompareKeyWithData: TListSortCompare)
                  : TAVLTreeNode
```

Visibility: public

Description: `FindKey` uses the specified `OnCompareKeyWithData` comparing function to find the `Key` pointer in the tree It returns the `TAVLTreeNode` instance that matches the `Data` pointer, or `Nil` if none is found.

See also: OnCompare (192), Find (186)

4.4.26 TAVLTree.FindNearestKey

Synopsis: Find nearest key for a data pointer.

Declaration:
```
function FindNearestKey(Key: Pointer;
                        const OnCompareKeyWithData: TListSortCompare)
                         : TAVLTreeNode
```

Visibility: public

Description: `FindNearestKey` attempts to find the nearest possible key in the tree using the `OnCompareKeyWithData` function. It returns the closest possible key in the tree.

Errors: None.

See also: TAVLTree.FindKey (186)

4.4.27 TAVLTree.FindSuccessor

Synopsis: Find successor to node.

Declaration:
```
function FindSuccessor(ANode: TAVLTreeNode) : TAVLTreeNode
```

Visibility: public

Description: `FindSuccessor` returns the successor to `ANode`: this is the leftmost node in the right subtree, or the leftmost node above the node `ANode`. This can of course be `Nil`.

This method is used when a node must be inserted at the rightmost position.

See also: TAVLTree.FindPrecessor (187), TAVLTree.MoveDataRightMost (184)

4.4.28 TAVLTree.FindPrecessor

Synopsis:

Declaration:
```
function FindPrecessor(ANode: TAVLTreeNode) : TAVLTreeNode
```

Visibility: public

Description: `FindPrecessor` returns the successor to `ANode`: this is the rightmost node in the left subtree, or the rightmost node above the node `ANode`. This can of course be `Nil`.

This method is used when a node must be inserted at the leftmost position.

See also: TAVLTree.FindSuccessor (187), TAVLTree.MoveDataLeftMost (184)

4.4.29 TAVLTree.FindLowest

Synopsis: Find the lowest (leftmost) node in the tree.

Declaration: `function FindLowest : TAVLTreeNode`

Visibility: public

Description: `FindLowest` returns the leftmost node in the tree, i.e. the node which is reached when descending from the rootnode via the left (**??**) subtrees.

See also: FindHighest (188)

4.4.30 TAVLTree.FindHighest

Synopsis: Find the highest (rightmost) node in the tree.

Declaration: `function FindHighest : TAVLTreeNode`

Visibility: public

Description: `FindHighest` returns the rightmost node in the tree, i.e. the node which is reached when descending from the rootnode via the Right (**??**) subtrees.

See also: FindLowest (188)

4.4.31 TAVLTree.FindNearest

Synopsis: Find the node closest to the data in the tree.

Declaration: `function FindNearest(Data: Pointer) : TAVLTreeNode`

Visibility: public

Description: `FindNearest` searches the node in the data tree that is closest to the specified `Data`. If `Data` appears in the tree, then its node is returned.

See also: FindHighest (188), FindLowest (188), Find (186), FindKey (186)

4.4.32 TAVLTree.FindPointer

Synopsis: Search for a data pointer.

Declaration: `function FindPointer(Data: Pointer) : TAVLTreeNode`

Visibility: public

Description: `FindPointer` searches for a node where the actual data pointer equals `Data`. This is a more fine search than find (186), where a custom compare function can be used.

The default OnCompare (192) compares the data pointers, so the default `Find` will return the same node as `FindPointer`

See also: TAVLTree.Find (186), TAVLTree.FindKey (186)

4.4.33 TAVLTree.FindLeftMost

Synopsis: Find the node most left to a specified data node.

Declaration:
```
function FindLeftMost(Data: Pointer) : TAVLTreeNode
```

Visibility: public

Description: `FindLeftMost` finds the node most left from the `Data` node. It starts at the preceding node for `Data` and tries to move as far right in the tree as possible.

This operation corresponds to finding the previous item in a list.

See also: TAVLTree.FindRightMost (189), TAVLTree.FindLeftMostKey (189), TAVLTree.FindRightMostKey (189)

4.4.34 TAVLTree.FindRightMost

Synopsis: Find the node most right to a specified node.

Declaration:
```
function FindRightMost(Data: Pointer) : TAVLTreeNode
```

Visibility: public

Description: `FindRightMost` finds the node most right from the `Data` node. It starts at the succeeding node for `Data` and tries to move as far left in the tree as possible.

This operation corresponds to finding the next item in a list.

See also: TAVLTree.FindLeftMost (189), TAVLTree.FindLeftMostKey (189), TAVLTree.FindRightMostKey (189)

4.4.35 TAVLTree.FindLeftMostKey

Synopsis: Find the node most left to a specified key node.

Declaration:
```
function FindLeftMostKey(Key: Pointer;
                         const OnCompareKeyWithData: TListSortCompare)
  : TAVLTreeNode
```

Visibility: public

Description: `FindLeftMostKey` finds the node most left from the node associated with `Key`. It starts at the preceding node for `Key` and tries to move as far left in the tree as possible.

See also: TAVLTree.FindLeftMost (189), TAVLTree.FindRightMost (189), TAVLTree.FindRightMostKey (189)

4.4.36 TAVLTree.FindRightMostKey

Synopsis: Find the node most right to a specified key node.

Declaration:
```
function FindRightMostKey(Key: Pointer;
                          const OnCompareKeyWithData: TListSortCompare)
  : TAVLTreeNode
```

Visibility: public

Description: `FindRightMostKey` finds the node most left from the node associated with `Key`. It starts at the succeeding node for `Key` and tries to move as far right in the tree as possible.

See also: TAVLTree.FindLeftMost (189), TAVLTree.FindRightMost (189), TAVLTree.FindLeftMostKey (189)

4.4.37 TAVLTree.FindLeftMostSameKey

Synopsis: Find the node most left to a specified node with the same data.

Declaration: `function FindLeftMostSameKey(ANode: TAVLTreeNode) : TAVLTreeNode`

Visibility: public

Description: `FindLefMostSameKey` finds the node most left from and with the same data as the specified node `ANode`.

See also: TAVLTree.FindLeftMost (189), TAVLTree.FindLeftMostKey (189), TAVLTree.FindRightMostSameKey (190)

4.4.38 TAVLTree.FindRightMostSameKey

Synopsis: Find the node most right of a specified node with the same data.

Declaration: `function FindRightMostSameKey(ANode: TAVLTreeNode) : TAVLTreeNode`

Visibility: public

Description: `FindRighMostSameKey` finds the node most right from and with the same data as the specified node `ANode`.

See also: TAVLTree.FindRightMost (189), TAVLTree.FindRightMostKey (189), TAVLTree.FindLeftMostSameKey (190)

4.4.39 TAVLTree.GetEnumerator

Synopsis: Get an enumerator for the tree.

Declaration: `function GetEnumerator : TAVLTreeNodeEnumerator`

Visibility: public

Description: `GetEnumerator` returns an instance of the standard tree node enumerator TAVLTreeNodeEnumerator (194).

See also: TAVLTreeNodeEnumerator (194)

4.4.40 TAVLTree.GetEnumeratorHighToLow

Synopsis: Return an enumerator that enumerates the tree in reversed order.

Declaration: `function GetEnumeratorHighToLow : TAVLTreeNodeEnumerator`

Visibility: public

Description: `GetEnumeratorHighToLow` returns an enumerated that traverses the tree in reversed order.

See also: TAVLTree.GetEnumerator (190)

4.4.41 TAVLTree.ConsistencyCheck

Synopsis: Check the consistency of the tree.

Declaration: `procedure ConsistencyCheck; Virtual`

Visibility: public

Description: `ConsistencyCheck` checks the correctness of the tree. It returns 0 if the tree is internally consistent, and a negative number if the tree contains an error somewhere.

-1 The Count property doesn't match the actual node count

-2 A left node does not point to the correct parent

-3 A left node is larger than parent node

-4 A right node does not point to the correct parent

-5 A right node is less than parent node

-6 The balance of a node is not calculated correctly

See also: TAVLTree.WriteReportToStream (191)

4.4.42 TAVLTree.WriteReportToStream

Synopsis: Write the contents of the tree consistency check to the stream.

Declaration: `procedure WriteReportToStream(s: TStream)`

Visibility: public

Description: `WriteReportToStream` writes a visual representation of the tree to the stream `S`. The total number of written bytes is returned in `StreamSize`. This method is only useful for debugging purposes.

See also: TAVLTree.ConsistencyCheck (191)

4.4.43 TAVLTree.NodeToReportStr

Synopsis: Create a textual dump of the tree.

Declaration: `function NodeToReportStr(aNode: TAVLTreeNode) : string; Virtual`

Visibility: public

Description: `NodeToReportStr` creates a textual representation of a node. It is called by TAVLTree.ReportAsString (191) for debugging purposes. It prints the data pointer as a hex value. Override this to create a human-readable representation of the data.

See also: TAVLTree.ReportAsString (191)

4.4.44 TAVLTree.ReportAsString

Synopsis: Return the tree report as a string.

Declaration: `function ReportAsString : string`

Visibility: public

Description: `ReportAsString` calls WriteReportToStream (191) and returns the stream data as a string.

See also: TAVLTree.WriteReportToStream (191)

4.4.45 TAVLTree.OnCompare

Synopsis: Compare function used when comparing nodes.

Declaration: `Property OnCompare : TListSortCompare`

Visibility: public

Access: Read,Write

Description: `OnCompare` is the comparing function used when the data of 2 nodes must be compared. By default, the function simply compares the 2 data pointers. A different function can be specified on creation.

See also: TAVLTree.Create (181)

4.4.46 TAVLTree.OnObjectCompare

Synopsis: Compare handler.

Declaration: `Property OnObjectCompare : TObjectSortCompare`

Visibility: public

Access: Read,Write

Description: `OnObjectCompare` is used to compare nodes. It is only used if TAVLTree.OnCompare (192) is not set.

See also: TAVLTree.OnCompare (192)

4.4.47 TAVLTree.NodeClass

Synopsis: Node class to create.

Declaration: `Property NodeClass : TAVLTreeNodeClass`

Visibility: public

Access: Read,Write

Description: `NodeClass` is the class of nodes to create when adding new nodes: TAVLTree.NewNode (182) will use this class when creating a new node. This can be set to a descendent class of TAVLTreeNode (193), but not if there are already nodes in the tree.

See also: TAVLTreeNode (193), TAVLTree.NewNode (182)

4.4.48 TAVLTree.Root

Synopsis: Root node of the tree.

Declaration: `Property Root : TAVLTreeNode`

Visibility: public

Access: Read

Description: `Root` is the root node of the tree. It should not be set explicitly, only use the Add (182), Delete (183), Remove (183), RemovePointer (184), or Clear (184) methods to manipulate the items in the tree.

See also: TAVLTree.Add (182), TAVLTree.Delete (183), TAVLTree.Remove (183), TAVLTree.RemovePointer (184), TAVLTree.Clear (184)

4.4.49 TAVLTree.Count

Synopsis: Number of nodes in the tree.

Declaration: `Property Count : SizeInt`

Visibility: public

Access: Read

Description: `Count` is the number of nodes in the tree.

4.5 TAVLTreeNode

4.5.1 Description

`TAVLTreeNode` represents a single node in the AVL tree. It contains references to the other nodes in the tree, and provides a Data (**??**) pointer which can be used to store the data, associated with the node.

See also: TAVLTree (179), TAVLTreeNode.Data (**??**)

4.5.2 Method overview

Page	Method	Description
194	Clear	Clears the node's data.
194	ConsistencyCheck	Check consistency of the node and below nodes.
194	GetCount	Get the number of nodes.
193	Precessor	Preceding node in the tree.
193	Successor	Succeeding node in the tree.
194	TreeDepth	Level of the node in the tree below.

4.5.3 TAVLTreeNode.Successor

Synopsis: Succeeding node in the tree.

Declaration: `function Successor : TAVLTreeNode`

Visibility: public

Description: `Successor` calculates and return the succeeding (right) node in the tree. For the last node, Nil is returned.

See also: TAVLTreeNode.Precessor (193)

4.5.4 TAVLTreeNode.Precessor

Synopsis: Preceding node in the tree.

Declaration: `function Precessor : TAVLTreeNode`

Visibility: public

Description: `Precessor` calculates and return the preceding (left) node in the tree. For the first node, Nil is returned.

See also: TAVLTreeNode.Successor (193)

4.5.5 TAVLTreeNode.Clear

Synopsis: Clears the node's data.

Declaration: `procedure Clear`

Visibility: public

Description: `Clear` clears all pointers and references in the node. It does not free the memory pointed to by these references.

4.5.6 TAVLTreeNode.TreeDepth

Synopsis: Level of the node in the tree below.

Declaration: `function TreeDepth : Integer`

Visibility: public

Description: `TreeDepth` is the height of the node: this is the largest height of the left or right nodes, plus 1. If no nodes appear below this node (`left` and `Right` are `Nil`), the depth is 1.

See also: Balance (**??**)

4.5.7 TAVLTreeNode.ConsistencyCheck

Synopsis: Check consistency of the node and below nodes.

Declaration: `procedure ConsistencyCheck(Tree: TAVLTree); Virtual`

Visibility: public

Description: `ConsistencyCheck` checks whether the node and nodes below are consistent, i.e. the nodes are still ordered correctly: left nodes are before right nodes.

Errors: If an inconsistency is detected, an exception is raised.

4.5.8 TAVLTreeNode.GetCount

Synopsis: Get the number of nodes.

Declaration: `function GetCount : SizeInt`

Visibility: public

Description: `GetCount` returns 1 plus the number of subnodes, if any.

Errors: None.

4.6 TAVLTreeNodeEnumerator

4.6.1 Description

`TAVLTreeNodeEnumerator` is a class which implements the enumerator interface for the TAVL-Tree (179). It enumerates all the nodes in the tree.

See also: TAVLTree (179)

4.6.2 Method overview

Page	Method	Description
195	Create	Create a new instance of TAVLTreeNodeEnumerator.
195	GetEnumerator	Returns the enumerator.
195	MoveNext	Move to next node in the tree.

4.6.3 Property overview

Page	Properties	Access	Description
195	Current	r	Current node in the tree.
196	LowToHigh	r	Should the enumerator return nodes from low to high or high to low.

4.6.4 TAVLTreeNodeEnumerator.Create

Synopsis: Create a new instance of TAVLTreeNodeEnumerator.

Declaration: `constructor Create(Tree: TAVLTree; aLowToHigh: Boolean)`

Visibility: public

Description: `Create` creates a new instance of `TAVLTreeNodeEnumerator` and saves the `Tree` argument for later use in the enumerator.

4.6.5 TAVLTreeNodeEnumerator.GetEnumerator

Synopsis: Returns the enumerator.

Declaration: `function GetEnumerator : TAVLTreeNodeEnumerator`

Visibility: public

Description: `GetEnumerator` returns `Self`..

4.6.6 TAVLTreeNodeEnumerator.MoveNext

Synopsis: Move to next node in the tree.

Declaration: `function MoveNext : Boolean`

Visibility: public

Description: `MoveNext` will return the lowest node in the tree to start with, and for all other calls returns the successor node of the current node with TAVLTree.FindSuccessor (187).

See also: TAVLTree.FindSuccessor (187)

4.6.7 TAVLTreeNodeEnumerator.Current

Synopsis: Current node in the tree.

Declaration: `Property Current : TAVLTreeNode`

Visibility: public

Access: Read

Description: `Current` is the current node in the enumeration.

See also: TAVLTreeNodeEnumerator.MoveNext (195)

4.6.8 TAVLTreeNodeEnumerator.LowToHigh

Synopsis: Should the enumerator return nodes from low to high or high to low.

Declaration: `Property LowToHigh : Boolean`

Visibility: public

Access: Read

Description: `LowToHigh` determines whether the tree is walked from low to high or high to low. It's value is set in the constructor and cannot be changed while enumerating the tree nodes.

See also: TAVLTreeNodeEnumerator.Create (195)

4.7 TAVLTreeNodeMemManager

4.7.1 Description

`TAVLTreeNodeMemManager` is an internal object used by the avl_tree unit. Normally, no instance of this object should be created: An instance is created by the unit initialization code, and freed when the unit is finalized.

See also: TAVLTreeNode (193), TAVLTree (179)

4.7.2 Method overview

Page	Method	Description
197	Clear	Frees all unused nodes.
197	Create	Create a new instance of `TAVLTreeNodeMemManager`.
197	Destroy	
196	DisposeNode	Return a node to the free list.
197	NewNode	Create a new `TAVLTreeNode` instance.

4.7.3 Property overview

Page	Properties	Access	Description
198	Count	r	Number of nodes in the list.
198	MaximumFreeNodeRatio	rw	Maximum amount of free nodes in the list.
198	MinimumFreeNode	rw	Minimum amount of free nodes to be kept.

4.7.4 TAVLTreeNodeMemManager.DisposeNode

Synopsis: Return a node to the free list.

Declaration: `procedure DisposeNode(ANode: TAVLTreeNode); Override`

Visibility: public

Description: `DisposeNode` is used to put the node `ANode` in the list of free nodes, or optionally destroy it if the free list is full. After a call to `DisposeNode`, `ANode` must be considered invalid.

See also: TAVLTreeNodeMemManager.NewNode (197)

4.7.5 TAVLTreeNodeMemManager.NewNode

Synopsis: Create a new `TAVLTreeNode` instance.

Declaration: `function NewNode : TAVLTreeNode;  Override`

Visibility: public

Description: `NewNode` returns a new TAVLTreeNode (193) instance. If there is a node in the free list, it is returned. If no more free nodes are present, a new node is created.

See also: TAVLTreeNodeMemManager.DisposeNode (196)

4.7.6 TAVLTreeNodeMemManager.Clear

Synopsis: Frees all unused nodes.

Declaration: `procedure Clear`

Visibility: public

Description: `Clear` removes all unused nodes from the list and frees them.

See also: TAVLTreeNodeMemManager.MinimumFreeNode (198), TAVLTreeNodeMemManager.MaximumFreeNodeRatio (198)

4.7.7 TAVLTreeNodeMemManager.Create

Synopsis: Create a new instance of `TAVLTreeNodeMemManager`.

Declaration: `constructor Create`

Visibility: public

Description: `Create` initializes a new instance of `TAVLTreeNodeMemManager`.

See also: TAVLTreeNodeMemManager.Destroy (197)

4.7.8 TAVLTreeNodeMemManager.Destroy

Synopsis:

Declaration: `destructor Destroy;  Override`

Visibility: public

Description: `Destroy` calls clear to clean up the free node list and then calls the inherited destroy.

See also: TAVLTreeNodeMemManager.Create (197)

4.7.9 TAVLTreeNodeMemManager.MinimumFreeNode

Synopsis: Minimum amount of free nodes to be kept.

Declaration: `Property MinimumFreeNode : SizeInt`

Visibility: public

Access: Read,Write

Description: `MinimumFreeNode` is the minimum amount of nodes that must be kept in the free nodes list.

See also: TAVLTreeNodeMemManager.MaximumFreeNodeRatio (198)

4.7.10 TAVLTreeNodeMemManager.MaximumFreeNodeRatio

Synopsis: Maximum amount of free nodes in the list.

Declaration: `Property MaximumFreeNodeRatio : SizeInt`

Visibility: public

Access: Read,Write

Description: `MaximumFreeNodeRatio` is the maximum amount of free nodes that should be kept in the list: if a node is disposed of, then the ratio of the free nodes versus the total amount of nodes is checked, and if it is less than the `MaximumFreeNodeRatio` ratio but larger than the minimum amount of free nodes, then the node is disposed of instead of added to the free list.

See also: TAVLTreeNodeMemManager.Count (198), TAVLTreeNodeMemManager.MinimumFreeNode (198)

4.7.11 TAVLTreeNodeMemManager.Count

Synopsis: Number of nodes in the list.

Declaration: `Property Count : SizeInt`

Visibility: public

Access: Read

Description: `Count` is the total number of nodes in the list, used or not.

See also: TAVLTreeNodeMemManager.MinimumFreeNode (198), TAVLTreeNodeMemManager.MaximumFreeNodeRatio (198)

4.8 TBaseAVLTreeNodeManager

4.8.1 Description

`TBaseAVLTreeNodeManager` is an abstract class from which a descendent can be created that manages creating and disposing of tree nodes (instances of TAVLTreeNode (193)) for a TAVLTree (179) tree instance. No instance of this class should be created, it is a purely abstract class. The default descendant of this class used by an `TAVLTree` instance is TAVLTreeNodeMemManager (196).

The TAVLTree.SetNodeManager (181) method can be used to set the node manager that a `TAVLTree` instance should use.

See also: TAVLTreeNodeMemManager (196), TAVLTree.SetNodeManager (181), TAVLTreeNode (193)

4.8.2 Method overview

Page	Method	Description
199	DisposeNode	Called when the AVL tree no longer needs node.
199	NewNode	Called when the AVL tree needs a new node.

4.8.3 TBaseAVLTreeNodeManager.DisposeNode

Synopsis: Called when the AVL tree no longer needs node.

Declaration: `procedure DisposeNode(ANode: TAVLTreeNode); Virtual; Abstract`

Visibility: public

Description: `DisposeNode` is called by TAVLTree (179) when it no longer needs a TAVLTreeNode (193) instance. The manager may decide to re-use the instance for later use instead of destroying it.

See also: TBaseAVLTreeNodeManager.NewNode (199), TAVLTree.Delete (183), TAVLTreeNode (193)

4.8.4 TBaseAVLTreeNodeManager.NewNode

Synopsis: Called when the AVL tree needs a new node.

Declaration: `function NewNode : TAVLTreeNode; Virtual; Abstract`

Visibility: public

Description: `NewNode` is called by TAVLTree (179) when it needs a new node in TAVLTree.Add (182). It must be implemented by descendants to return a new TAVLTreeNode (193) instance.

See also: TBaseAVLTreeNodeManager.DisposeNode (199), TAVLTree.Add (182), TAVLTreeNode (193)

Chapter 5

Reference for unit 'base64'

5.1 Used units

Table 5.1: Used units by unit 'base64'

Name	Page
Classes	??
System	??
sysutils	??

5.2 Overview

base64 implements base64 encoding (as used for instance in MIME encoding) based on streams. It implements 2 streams which encode or decode anything written or read from it. The source or the destination of the encoded data is another stream. 2 classes are implemented for this: TBase64EncodingStream (204) for encoding, and TBase64DecodingStream (202) for decoding.

The streams are designed as plug-in streams, which can be placed between other streams, to provide base64 encoding and decoding on-the-fly...

5.3 Constants, types and variables

5.3.1 Types

```
TBase64DecodingMode = (bdmStrict,bdmMIME)
```

Table 5.2: Enumeration values for type TBase64DecodingMode

Value	Explanation
bdmMIME	MIME encoding.
bdmStrict	Strict encoding.

TBase64DecodingMode determines the decoding algorithm used by TBase64DecodingStream (202). There are 2 modes:

bdmStrict Strict mode, which follows RFC3548 and rejects any characters outside of base64 alphabet. In this mode only up to two '=' characters are accepted at the end. It requires the input to have a Size being a multiple of 4, otherwise an EBase64DecodingException (201) exception is raised.

bdmMime MIME mode, which follows RFC2045 and ignores any characters outside of base64 alphabet. In this mode any '=' is seen as the end of string, it handles apparently truncated input streams gracefully.

5.4 Procedures and functions

5.4.1 DecodeStringBase64

Synopsis: Decodes a Base64 encoded string and returns the decoded data as a string.

Declaration:
```
function DecodeStringBase64(const s: AnsiString; strict: Boolean)
                                  : AnsiString
```

Visibility: default

Description: DecodeStringBase64 decodes the string s (containing Base 64 encoded data) returns the decoded data as a string. It uses a TBase64DecodingStream (202) to do this. The Strict parameter is passed on to the constructor as bdmStrict or bdmMIME

See also: DecodeStringBase64 (201), TBase64DecodingStream (202)

5.4.2 EncodeStringBase64

Synopsis: Encode a string with Base64 encoding and return the result as a string.

Declaration:
```
function EncodeStringBase64(const s: AnsiString) : AnsiString
```

Visibility: default

Description: EncodeStringBase64 encodes the string s using Base 64 encoding and returns the result. It uses a TBase64EncodingStream (204) to do this.

See also: DecodeStringBase64 (201), TBase64EncodingStream (204)

5.5 EBase64DecodingException

5.5.1 Description

EBase64DecodeException is raised when the stream contains errors against the encoding format. Whether or not this exception is raised depends on the mode in which the stream is decoded.

5.6 TBase64DecodingStream

5.6.1 Description

`TBase64DecodingStream` can be used to read data from a stream (the source stream) that contains Base64 encoded data. The data is read and decoded on-the-fly.

The decoding stream is read-only, and provides a limited forward-seek capability.

See also: TBase64EncodingStream (204)

5.6.2 Method overview

Page	Method	Description
202	Create	Create a new instance of the `TBase64DecodingStream` class.
203	Read	Read and decrypt data from the source stream.
202	Reset	Reset the stream.
203	Seek	Set stream position.

5.6.3 Property overview

Page	Properties	Access	Description
203	EOF	r	
203	Mode	rw	Decoding mode.

5.6.4 TBase64DecodingStream.Create

Synopsis: Create a new instance of the `TBase64DecodingStream` class.

Declaration:
```
constructor Create(ASource: TStream)
constructor Create(ASource: TStream; AMode: TBase64DecodingMode)
```

Visibility: public

Description: `Create` creates a new instance of the `TBase64DecodingStream` class. It stores the source stream `ASource` for reading the data from.

The optional `AMode` parameter determines the mode in which the decoding will be done. If omitted, `bdmMIME` is used.

See also: TBase64EncodingStream (204), TBase64DecodingMode (200)

5.6.5 TBase64DecodingStream.Reset

Synopsis: Reset the stream.

Declaration:
```
procedure Reset
```

Visibility: public

Description: `Reset` resets the data as if it was again on the start of the decoding stream.

Errors: None.

See also: TBase64DecodingStream.EOF (203), TBase64DecodingStream.Read (203)

5.6.6 TBase64DecodingStream.Read

Synopsis: Read and decrypt data from the source stream.

Declaration: `function Read(var Buffer; Count: LongInt) : LongInt;  Override`

Visibility: public

Description: `Read` reads encrypted data from the source stream and stores this data in `Buffer`. At most `Count` bytes will be stored in the buffer, but more bytes will be read from the source stream: the encoding algorithm multiplies the number of bytes.

The function returns the number of bytes stored in the buffer.

Errors: If an error occurs during the read from the source stream, an exception may occur.

See also: TBase64DecodingStream.Seek (203), TStream.Read (**??**)

5.6.7 TBase64DecodingStream.Seek

Synopsis: Set stream position.

Declaration: `function Seek(Offset: LongInt; Origin: Word) : LongInt;  Override`

Visibility: public

Description: `Seek` sets the position of the stream. In the `TBase64DecodingStream` class, the `seek` operation is forward only, it does not support backward seeks. The forward seek is emulated by reading and discarding data till the desired position is reached.

For an explanation of the parameters, see TStream.Seek (**??**)

Errors: In case of an unsupported operation, an `EStreamError` exception is raised.

See also: TBase64DecodingStream.Read (203), TBase64EncodingStream.Seek (205), TStream.Seek (**??**)

5.6.8 TBase64DecodingStream.EOF

Synopsis:

Declaration: `Property EOF : Boolean`

Visibility: public

Access: Read

Description:

5.6.9 TBase64DecodingStream.Mode

Synopsis: Decoding mode.

Declaration: `Property Mode : TBase64DecodingMode`

Visibility: public

Access: Read,Write

Description: `Mode` is the mode in which the stream is read. It can be set when creating the stream or at any time afterwards.

See also: TBase64DecodingStream (202)

5.7 TBase64EncodingStream

5.7.1 Description

`TBase64EncodingStream` can be used to encode data using the base64 algorithm. At creation time, a destination stream is specified. Any data written to the `TBase64EncodingStream` instance will be base64 encoded, and subsequently written to the destination stream.

The `TBase64EncodingStream` stream is a write-only stream. Obviously it is also not seekable. It is meant to be included in a chain of streams.

By the nature of base64 encoding, when a buffer is written to the stream, the output stream does not yet contain all output: input must be a multiple of 3. In order to be sure that the output contains all encoded bytes, the Flush (204) method can be used. The destructor will automatically call `Flush`, so all data is written to the destination stream when the decodes is destroyed.

See also: TBase64DecodingStream (202)

5.7.2 Method overview

Page	Method	Description
204	Destroy	Remove a `TBase64EncodingStream` instance from memory.
204	Flush	Flush the remaining bytes to the output stream.
205	Seek	Position the stream.
205	Write	Write data to the stream.

5.7.3 TBase64EncodingStream.Destroy

Synopsis: Remove a `TBase64EncodingStream` instance from memory.

Declaration:
```
destructor Destroy;  Override
```

Visibility: public

Description: `Destroy` flushes any remaining output and then removes the `TBase64EncodingStream` instance from memory by calling the inherited destructor.

Errors: An exception may be raised if the destination stream no longer exists or is closed.

5.7.4 TBase64EncodingStream.Flush

Synopsis: Flush the remaining bytes to the output stream.

Declaration:
```
function Flush : Boolean
```

Visibility: public

Description: `Flush` writes the remaining bytes from the internal encoding buffer to the output stream and pads the output with "=" signs. It returns `True` if padding was necessary, and `False` if not.

See also: TBase64EncodingStream.Destroy (204)

5.7.5 TBase64EncodingStream.Write

Synopsis: Write data to the stream.

Declaration: `function Write(const Buffer; Count: LongInt) : LongInt;  Override`

Visibility: public

Description: `Write` encodes `Count` bytes from `Buffer` using the Base64 mechanism, and then writes the encoded data to the destination stream. It returns the number of bytes from `Buffer` that were actually written. Note that this is not the number of bytes written to the destination stream: the base64 mechanism writes more bytes to the destination stream.

Errors: If there is an error writing to the destination stream, an error may occur.

See also: TBase64EncodingStream.Seek (205), TStream.Write (**??**)

5.7.6 TBase64EncodingStream.Seek

Synopsis: Position the stream.

Declaration: `function Seek(Offset: LongInt; Origin: Word) : LongInt;  Override`

Visibility: public

Description: `Seek` always raises an `EStreamError` exception unless the arguments it received it don't change the current file pointer position. The encryption stream is not seekable.

Errors: An `EStreamError` error is raised.

See also: TBase64EncodingStream.Write (205), TStream.Seek (**??**)

Chapter 6

Reference for unit 'BlowFish'

6.1 Used units

Table 6.1: Used units by unit 'BlowFish'

Name	Page
Classes	**??**
System	**??**
sysutils	**??**

6.2 Overview

The BlowFish implements a class TBlowFish (207) to handle Blowfish encryption/decryption of memory buffers, and 2 TStream (**??**) descendents TBlowFishDeCryptStream (208) which decrypts any data that is read from it on the fly, as well as TBlowFishEnCryptStream (209) which encrypts the data that is written to it on the fly.

6.3 Constants, types and variables

6.3.1 Constants

```
BFRounds = 16
```

Number of rounds in Blowfish encryption.

6.3.2 Types

```
PBlowFishKey = ^TBlowFishKey
```

`PBlowFishKey` is a simple pointer to a TBlowFishKey (207) array.

```
TBFBlock = Array[0..1] of LongInt
```

TBFBlock is the basic data structure used by the encrypting/decrypting routines in TBlowFish (207), TBlowFishDeCryptStream (208) and TBlowFishEnCryptStream (209). It is the basic encryption/decryption block for all encrypting/decrypting: all encrypting/decrypting happens on a TBFBlock structure.

```
TBlowFishKey = Array[0..55] of Byte
```

TBlowFishKey is a data structure which keeps the encryption or decryption key for the TBlowFish (207), TBlowFishDeCryptStream (208) and TBlowFishEnCryptStream (209) classes. It should be filled with the encryption key and passed to the constructor of one of these classes.

6.4 EBlowFishError

6.4.1 Description

EBlowFishError is used by the TBlowFishStream (211), TBlowFishEncryptStream (209) and TBlowFishDecryptStream (208) classes to report errors.

See also: TBlowFishStream (211), TBlowFishEncryptStream (209), TBlowFishDecryptStream (208)

6.5 TBlowFish

6.5.1 Description

TBlowFish is a simple class that can be used to encrypt/decrypt a single TBFBlock (207) data block with the Encrypt (208) and Decrypt (208) calls. It is used internally by the TBlowFishEnCryptStream (209) and TBlowFishDeCryptStream (208) classes to encrypt or decrypt the actual data.

See also: TBlowFishEnCryptStream (209), TBlowFishDeCryptStream (208)

6.5.2 Method overview

Page	Method	Description
207	Create	Create a new instance of the TBlowFish class.
208	Decrypt	Decrypt a block.
208	Encrypt	Encrypt a block.

6.5.3 TBlowFish.Create

Synopsis: Create a new instance of the TBlowFish class.

Declaration: `constructor Create(Key: TBlowFishKey; KeySize: Integer)`

Visibility: public

Description: Create initializes a new instance of the TBlowFish class: it stores the key Key in the internal data structures so it can be used in later calls to Encrypt (208) and Decrypt (208).

See also: Encrypt (208), Decrypt (208)

6.5.4 TBlowFish.Encrypt

Synopsis: Encrypt a block.

Declaration: `procedure Encrypt(var Block: TBFBlock)`

Visibility: public

Description: `Encrypt` encrypts the data in `Block` (always 8 bytes) using the key (207) specified when the `TBlowFish` instance was created.

See also: TBlowFishKey (207), Decrypt (208), Create (207)

6.5.5 TBlowFish.Decrypt

Synopsis: Decrypt a block.

Declaration: `procedure Decrypt(var Block: TBFBlock)`

Visibility: public

Description: `ncrypt` decrypts the data in `Block` (always 8 bytes) using the key (207) specified when the `TBlowFish` instance was created. The data must have been encrypted with the same key and the Encrypt (208) call.

See also: TBlowFishKey (207), Encrypt (208), Create (207)

6.6 TBlowFishDeCryptStream

6.6.1 Description

The `TBlowFishDecryptStream` provides On-the-fly Blowfish decryption: all data that is read from the source stream is decrypted before it is placed in the output buffer. The source stream must be specified when the `TBlowFishDecryptStream` instance is created. The Decryption key must also be created when the stream instance is created, and must be the same key as the one used when encrypting the data.

This is a read-only stream: it is seekable only in a forward direction, and data can only be read from it, writing is not possible. For writing data so it is encrypted, the TBlowFishEncryptStream (209) stream must be used.

See also: Create (211), TBlowFishEncryptStream (209)

6.6.2 Method overview

Page	Method	Description
208	Create	Constructor for the class instance.
209	Read	Read data from the stream.
209	Seek	Set the stream position.

6.6.3 TBlowFishDeCryptStream.Create

Synopsis: Constructor for the class instance.

Declaration: `constructor Create(AKey: TBlowFishKey; AKeySize: Byte; Dest: TStream) ; Override`

Visibility: public

Description: `Create` is the overridden constructor for the class instance. It calls the inherited constructor on entry using the values in `AKey`, `AKeySize`, and `Dest` as arguments.

Create sets the value in an internal member to the current position in Dest. It acts as the relative origin for the TStream instance, and is used in the `Seek` method.

See also: TBlowFishDeCryptStream.Read (209), TBlowFishDeCryptStream.Seek (209), TBlowFishEnCryptStream.Write (210)

6.6.4 TBlowFishDeCryptStream.Read

Synopsis: Read data from the stream.

Declaration:
```
function Read(var Buffer; Count: LongInt) : LongInt;  Override
```

Visibility: public

Description: `Read` reads `Count` bytes from the source stream, decrypts them using the key provided when the `TBlowFishDeCryptStream` instance was created, and writes the decrypted data to `Buffer`.

See also: Create (211), TBlowFishEncryptStream (209)

6.6.5 TBlowFishDeCryptStream.Seek

Synopsis: Set the stream position.

Declaration:
```
function Seek(const Offset: Int64; Origin: TSeekOrigin) : Int64
                  ;  Override
```

Visibility: public

Description: `Seek` emulates a forward seek by reading and discarding data. The discarded data is lost. Since it is a forward seek, this means that only `soFromCurrent` can be specified for `Origin` with a positive (or zero) `Offset` value. All other values will result in an exception. The function returns the new position in the stream.

Errors: If any other combination of `Offset` and `Origin` than the allowed combination is specified, then an EBlowFishError (207) exception will be raised.

See also: Read (209), EBlowFishError (207)

6.7 TBlowFishEncryptStream

6.7.1 Description

The `TBlowFishEncryptStream` provides On-the-fly Blowfish encryption: all data that is written to it is encrypted and then written to a destination stream, which must be specified when the `TBlowFishEncryptStream` instance is created. The encryption key must also be created when the stream instance is created.

This is a write-only stream: it is not seekable, and data can only be written to it, reading is not possible. For reading encrypted data, the TBlowFishDecryptStream (208) stream must be used.

See also: Create (211), TBlowFishDecryptStream (208)

6.7.2 Method overview

Page	Method	Description
210	Destroy	Free the `TBlowFishEncryptStream`.
211	Flush	Flush the encryption buffer.
210	Seek	Set the position in the stream.
210	Write	Write data to the stream.

6.7.3 TBlowFishEncryptStream.Destroy

Synopsis: Free the `TBlowFishEncryptStream`.

Declaration:
```
destructor Destroy;  Override
```

Visibility: public

Description: `Destroy` flushes the encryption buffer, and writes it to the destination stream. After that the `Inherited` destructor is called to clean up the `TBlowFishEncryptStream` instance.

See also: Flush (211), Create (211)

6.7.4 TBlowFishEncryptStream.Write

Synopsis: Write data to the stream.

Declaration:
```
function Write(const Buffer; Count: LongInt) : LongInt;  Override
```

Visibility: public

Description: `Write` will encrypt and write `Count` bytes from `Buffer` to the destination stream. The function returns the actual number of bytes written. The data is not encrypted in-place, but placed in a special buffer for encryption.

Data is always written 4 bytes at a time, since this is the amount of bytes required by the Blowfish algorithm. If no multiple of 4 was written to the destination stream, the Flush (211) mechanism can be used to write the remaining bytes.

See also: TBlowFishEncryptStream.Flush (211)

6.7.5 TBlowFishEncryptStream.Seek

Synopsis: Set the position in the stream.

Declaration:
```
function Seek(const Offset: Int64; Origin: TSeekOrigin) : Int64
                  ;  Override
```

Visibility: public

Description: `Read` will raise an `EBlowFishError` exception: `TBlowFishEncryptStream` is a write-only stream, and cannot be positioned.

Errors: Calling this function always results in an EBlowFishError (207) exception.

See also: TBlowFishEncryptStream.Write (210)

6.7.6 TBlowFishEncryptStream.Flush

Synopsis: Flush the encryption buffer.

Declaration:
```
procedure Flush
```

Visibility: public

Description: `Flush` writes the remaining data in the encryption buffer to the destination stream.

For efficiency, data is always written 4 bytes at a time, since this is the amount of bytes required by the Blowfish algorithm. If no multiple of 4 was written to the destination stream, the `Flush` mechanism can be used to write the remaining bytes.

`Flush` is called automatically when the stream is destroyed, so there is no need to call it after all data was written and the stream is no longer needed.

See also: Write (210), TBFBlock (207)

6.8 TBlowFishStream

6.8.1 Description

`TBlowFishStream` is an abstract class which is used as a parent class for TBlowFishEncryptStream (209) and TBlowFishDecryptStream (208). It simply provides a constructor and storage for a TBlowFish (207) instance and for the source or destination stream.

Do not create an instance of `TBlowFishStream` directly. Instead create one of the descendent classes `TBlowFishEncryptStream` or `TBlowFishDecryptStream`.

See also: TBlowFishEncryptStream (209), TBlowFishDecryptStream (208), TBlowFish (207)

6.8.2 Method overview

Page	Method	Description
211	Create	Create a new instance of the `TBlowFishStream` class.
212	Destroy	Destroy the `TBlowFishStream` instance.

6.8.3 Property overview

Page	Properties	Access	Description
212	BlowFish	r	Blowfish instance used when encrypting/decrypting.

6.8.4 TBlowFishStream.Create

Synopsis: Create a new instance of the `TBlowFishStream` class.

Declaration:
```
constructor Create(AKey: TBlowFishKey; AKeySize: Byte; Dest: TStream)
                   ; Virtual; Overload
constructor Create(const KeyPhrase: string; Dest: TStream); Overload
```

Visibility: public

Description: `Create` initializes a new instance of `TBlowFishStream`, and creates an internal instance of TBlowFish (207) using `AKey` and `AKeySize`. The `Dest` stream is stored so the descendent classes can refer to it.

Do not create an instance of `TBlowFishStream` directly. Instead create one of the descendent classes `TBlowFishEncryptStream` or `TBlowFishDecryptStream`.

The overloaded version with the `KeyPhrase` string argument is used for easy access: it computes the Blowfish key from the given string.

See also: TBlowFishEncryptStream (209), TBlowFishDecryptStream (208), TBlowFish (207)

6.8.5 TBlowFishStream.Destroy

Synopsis: Destroy the `TBlowFishStream` instance.

Declaration: `destructor Destroy;  Override`

Visibility: public

Description: `Destroy` cleans up the internal TBlowFish (207) instance.

See also: Create (211), TBlowFish (207)

6.8.6 TBlowFishStream.BlowFish

Synopsis: Blowfish instance used when encrypting/decrypting.

Declaration: `Property BlowFish : TBlowFish`

Visibility: public

Access: Read

Description: `BlowFish` is the TBlowFish (207) instance which is created when the `TBlowFishStream` class is initialized. Normally it should not be used directly, it's intended for access by the descencent classes TBlowFishEncryptStream (209) and TBlowFishDecryptStream (208).

See also: TBlowFishEncryptStream (209), TBlowFishDecryptStream (208), TBlowFish (207)

Chapter 7

Reference for unit 'BufDataset'

7.1 Used units

Table 7.1: Used units by unit 'BufDataset'

Name	Page
bufdataset_parser	??
Classes	??
DB	438
System	??
sysutils	??

7.2 Overview

Implements a database-aware buffered dataset.

Original Authors: Joost van der Sluis and members of the Free Pascal development team.

7.3 Constants, types and variables

7.3.1 Types

```
PBlobBuffer = ^TBlobBuffer
```

`PBlobBuffer` is a pointer to the `TBlobBuffer` type. PBlobBuffer is used to allocate and access the buffers for Blob fields (Memo or Graphic field types). PBlobBuffer is used in TBufBlobField, TBufBlobStream, and in TBufDataset methods which perform record buffer management.

```
PBufBlobField = ^TBufBlobField
```

Pointer to a TBufBlobField type.

```
PBufBookmark = ^TBufBookmark
```

Pointer to the TBufBookmark type.

```
PBufRecLinkItem = ^TBufRecLinkItem
```

Pointer to a TBufRecLinkItem type.

```
TCompareFunc = function(subValue: pointer; aValue: pointer;
  size: Integer; options: TLocateOptions) :
  Int64
```

TCompareFunc is an Int64 function type used to compare pointers using the specified locate option values. TCompareFunc is used in the implementation of the TDBCompareRec type.

```
TDataPacketFormat = (dfBinary,dfXML,dfXMLUTF8,dfAny,dfDefault)
```

Table 7.2: Enumeration values for type TDataPacketFormat

Value	Explanation
dfAny	Data packets can be in any supported format (detected by the packet reader).
dfBinary	Data packets are in binary format.
dfDefault	Use the default XML format. Same as using dfAny.
dfXML	Data packets are in XML format.
dfXMLUTF8	Data packets are in UTF-8-encoded XML format.

Indicates the format used for data packets in TBufDataset.

```
TDataPacketHandlerClass = Class of TDataPacketHandler
```

```
TDataPacketReader = TDataPacketHandler
```

TDataPacketReader is a TObject descendant which implements a data packet reader for TBufDataset. TDataPacketReader is a concept borrowed from TClientDatset in Delphi. Data packets are used to access the values in its dataset. Data packets can contain field definitions or record data. Data packets can store their values in binary format or as an XML document.

TDataPacketReader provides properties and methods that allow access to the field definitions, field values, and row states for record buffers in its dataset. Methods are provided to load and save field definitions or record data using a stream. Methods are provided to recognize the storage format for the data packets in the stream.

Many of the methods in TDataPacketReader are declared as virtual or abstract. They are implemented in descendent classes that use a specific data packet format, such as TFpcBinaryDatapacketReader or TXMLDatapacketReader (in xmldatapacketreader.pp).

```
TDataPacketReaderClass = TDataPacketHandlerClass
```

Class type used to create data packet readers for TBufDataset.

```
TDBCompareStruct = Array of TDBCompareRec
```

TDBCompareStruct is an array of TDBCompareRec type used to implement a structure to compare values for a group of fields in TBufDataset. TDBCompareStruct is used in the implementation of TBufIndex and its descendent classes.

```
TFpcBinaryDatapacketReader = TFpcBinaryDatapacketHandler
```

TFpcBinaryDatapacketReader is a TDatapacketReader descendant that implements a data packet reader using binary data packets. Binary data packets use the following layout:

Table 7.3:

Section	Name	Length	Description
Header	Identification	13 bytes	Contains the value 'B
	Version	1 byte	Version number
	Field Definition Count	2 bytes	Column Layout for fie
Field Definitions	Field Name Length	2 bytes	Section occurs the nur
	Field Name	See above	Name of the field in th
	Display Name Length	2 bytes	
	Display Name	See above	Caption for the field
	Data Type	2 bytes	
	Read-Only Attribute	1 byte	0=read/write, 1=read-
Parameters	AutoInc Value	4 bytes	Integer value for the a
Row Header	Row Marker	1 byte	Contains the value $f
	Row State	1 byte	0=original, 1=deleted
	Update Order	4 bytes	
	Null Bitmap	Variable length.	1 byte required for eig
Row Data	Optional Field Length	4 bytes	Variable-length fields
	Field Data	Bytes values for the field. See Optional Field Length.	Occurs the number of

```
TRecordsUpdateBuffer = Array of TRecUpdateBuffer
```

TRecordsUpdateBuffer is an array of TRecUpdateBuffer type used to implement an array of update buffers for records in TBufDataset.

```
TResolverErrorEvent = procedure(Sender: TObject;
  DataSet: TCustomBufDataset;
  E: EUpdateError;
                                  UpdateKind
  : TUpdateKind;
                                  var Response: TResolverResponse
  )
                                   of object
```

TResolverErrorEvent is an object procedure which defines an event handler signalled when an error occurs while updating a record in TBufDataset. TResolverErrorEvent is the type used for the TCustomBufDataset.OnUpdateError property. Applications should create a routine which uses the signature for the type to handle the event notification. The routine should update the Response argument to indicate the action taken for the event notification.

```
TRowState = Set of TRowStateValue
```

TRowState is a set type used to store values from the TRowStateValue enumeration. TRowState is used to indicate the state for records loaded and saved using the data packet handler in TBufDataset.

```
TRowStateValue = (rsvOriginal,rsvDeleted,rsvInserted,rsvUpdated,
  rsvDetailUpdates)
```

Table 7.4: Enumeration values for type TRowStateValue

Value	Explanation
rsvDeleted	Buffer is for a record that will be deleted.
rsvDetailUpdates	Buffer is for updates to a detail (child) dataset.
rsvInserted	Buffer is for a record that will be inserted.
rsvOriginal	Buffer contains the original value(s).
rsvUpdated	Buffer is for a record that will be updated.

`TRowStateValue` is an enumeration type with values that indicate the state for field values in the row. Values in the enumeration indicate how fields in an update buffer differ from the fields in the original record data. Values from TRowStateValue are stored in the TRowState type, and used in data packet readers for TBufDataset (TDataPacketReader).

7.4 Procedures and functions

7.4.1 RegisterDatapacketReader

Synopsis: Registers a data packet reader using a specific format for TBufDataset.

Declaration:
```
procedure RegisterDatapacketReader
                                        (ADatapacketReaderClass: TDataPacketReaderClass;
                                        AFormat: TDataPacketFormat)
```

Visibility: default

Description: `RegisterDatapacketReader` is a procedure which registers a class type needed to create data packet readers for a specific format. `ADatapacketReaderClass` is the class type used to create new instances of `TDataPacketReader`. `AFormat` is a `TDataPacketFormat` enumeration value that indicates the format supported for data packets in the reader class.

RegisterDatapacketReader accesses a unit variable used to store an array of registered data packet reader classes and their associated formats. RegisterDatapacketReader increments the size of the array variable, and stores the values in ADatapacketReaderClass and AFormat in the TDatapacketReaderRegistration type. It does not perform any checks for duplicate entries, so the procedure should be called exactly once for the class type and format combination.

RegisterDatapacketReader is normally used in the initialization section of a unit that implements a TDataPacketReader descendant for a specific storage format, such as: `xmldatapacketreader.pp`.

See also: TDataPacketReader (214), TDataPacketFormat (214), TXMLDatapacketReader (438)

7.5 TApplyRecUpdateResult

```
TApplyRecUpdateResult = record
  HadError : Boolean;
  Response :
  TResolverResponse;
```

```
  Async : Boolean;
end
```

7.6 TBlobBuffer

```
TBlobBuffer = record
  FieldNo : Integer;
  OrgBufID : Integer;
  Buffer : pointer;
  Size : PtrInt;
end
```

`TBlobBuffer` is a record type used to represent a buffer allocated for a Blob field. Pointers to TBlobBuffer (PBlobBuffer) are used in the implementation of TBufBlobStream and TBufDataset.

7.7 TBufBlobField

```
TBufBlobField = record
  ConnBlobBuffer : Array[0..11] of Byte;
  BlobBuffer : PBlobBuffer;
end
```

Implements a Blob field for TBufDataset.

7.8 TBufBookmark

```
TBufBookmark = record
  BookmarkData : PBufRecLinkItem;
  BookmarkInt
  : Integer;
  BookmarkFlag : TBookmarkFlag;
end
```

TBufBookmark is a record type used to implement a Bookmark for TBufDataset.

7.9 TBufRecLinkItem

```
TBufRecLinkItem = record
  prior : PBufRecLinkItem;
  next : PBufRecLinkItem
  ;
end
```

Stores bookmarks to the previous and next records in a linked list.

7.10 TDBCompareRec

```
TDBCompareRec = record
  CompareFunc : TCompareFunc;
  Off : PtrInt
  ;
  NullBOff : PtrInt;
  FieldInd : LongInt;
  Size : Integer;
  Options
  : TLocateOptions;
  Desc : Boolean;
end
```

`TDBCompareRec` is a record type which implements a structure used to compare buffers for fields in TBufDataset. TDBCompareRec is used in the implementation of the TDBCompareStruct type.

7.11 TRecUpdateBuffer

```
TRecUpdateBuffer = record
  Processing : Boolean;
  UpdateKind : TUpdateKind
  ;
  BookmarkData : TBufBookmark;
  NextBookmarkData : TBufBookmark
  ;
  OldValuesBuffer : TRecordBuffer;
end
```

Implements a buffer used for record updates in TBufDataset.

7.12 TArrayBufIndex

7.12.1 Description

`TArrayBufIndex` is a `TBufIndex` descendant that implements an index using an array of record buffers. TArrayBufIndex uses an internal member with an array of pointers to the TRecordBuffer data type used for record buffers in the index. TArrayBufIndex provides support for Bookmarks in the dataset using TBufBookmark.

See also: TBufIndex (229), TRecordBuffer (213), TBufBookmark (217)

7.12.2 Method overview

Page	Method	Description
222	AddRecord	
222	BeginUpdate	
221	CanScrollForward	
219	Create	
221	DoScrollForward	
222	EndUpdate	
219	GetCurrent	
221	GotoBookmark	
221	InitialiseIndex	
221	InitialiseSpareRecord	
222	InsertRecordBeforeCurrentRecord	
222	ReleaseSpareRecord	
222	RemoveRecordFromIndex	
220	RestoreCurrentRecord	
219	ScrollBackward	
220	ScrollFirst	
219	ScrollForward	
220	ScrollLast	
220	SetToFirstRecord	
220	SetToLastRecord	
221	StoreCurrentRecIntoBookmark	
220	StoreCurrentRecord	
221	StoreSpareRecIntoBookmark	

7.12.3 TArrayBufIndex.Create

Synopsis:

Declaration: `constructor Create(const ADataset: TCustomBufDataset);  Override`

Visibility: public

7.12.4 TArrayBufIndex.ScrollBackward

Synopsis:

Declaration: `function ScrollBackward : TGetResult;  Override`

Visibility: public

7.12.5 TArrayBufIndex.ScrollForward

Synopsis:

Declaration: `function ScrollForward : TGetResult;  Override`

Visibility: public

7.12.6 TArrayBufIndex.GetCurrent

Synopsis:

Declaration: `function GetCurrent : TGetResult;  Override`

Visibility: public

7.12.7 TArrayBufIndex.ScrollFirst

Synopsis:

Declaration: `function ScrollFirst : TGetResult;  Override`

Visibility: public

7.12.8 TArrayBufIndex.ScrollLast

Synopsis:

Declaration: `procedure ScrollLast;  Override`

Visibility: public

7.12.9 TArrayBufIndex.SetToFirstRecord

Synopsis:

Declaration: `procedure SetToFirstRecord;  Override`

Visibility: public

7.12.10 TArrayBufIndex.SetToLastRecord

Synopsis:

Declaration: `procedure SetToLastRecord;  Override`

Visibility: public

7.12.11 TArrayBufIndex.StoreCurrentRecord

Synopsis:

Declaration: `procedure StoreCurrentRecord;  Override`

Visibility: public

7.12.12 TArrayBufIndex.RestoreCurrentRecord

Synopsis:

Declaration: `procedure RestoreCurrentRecord;  Override`

Visibility: public

7.12.13 TArrayBufIndex.CanScrollForward

Synopsis:

Declaration: `function CanScrollForward : Boolean;  Override`

Visibility: public

7.12.14 TArrayBufIndex.DoScrollForward

Synopsis:

Declaration: `procedure DoScrollForward;  Override`

Visibility: public

7.12.15 TArrayBufIndex.StoreCurrentRecIntoBookmark

Synopsis:

Declaration:
```
procedure StoreCurrentRecIntoBookmark(const ABookmark: PBufBookmark)
                                      ;  Override
```

Visibility: public

7.12.16 TArrayBufIndex.StoreSpareRecIntoBookmark

Synopsis:

Declaration:
```
procedure StoreSpareRecIntoBookmark(const ABookmark: PBufBookmark)
                                    ;  Override
```

Visibility: public

7.12.17 TArrayBufIndex.GotoBookmark

Synopsis:

Declaration: `procedure GotoBookmark(const ABookmark: PBufBookmark);  Override`

Visibility: public

7.12.18 TArrayBufIndex.InitialiseIndex

Synopsis:

Declaration: `procedure InitialiseIndex;  Override`

Visibility: public

7.12.19 TArrayBufIndex.InitialiseSpareRecord

Synopsis:

Declaration:
```
procedure InitialiseSpareRecord(const ASpareRecord: TRecordBuffer)
                                ;  Override
```

Visibility: public

7.12.20 TArrayBufIndex.ReleaseSpareRecord

Synopsis:

Declaration: `procedure ReleaseSpareRecord;  Override`

Visibility: public

7.12.21 TArrayBufIndex.BeginUpdate

Synopsis:

Declaration: `procedure BeginUpdate;  Override`

Visibility: public

7.12.22 TArrayBufIndex.AddRecord

Synopsis:

Declaration: `procedure AddRecord;  Override`

Visibility: public

7.12.23 TArrayBufIndex.InsertRecordBeforeCurrentRecord

Synopsis:

Declaration:
```
procedure InsertRecordBeforeCurrentRecord(const ARecord: TRecordBuffer)
                                                ;  Override
```

Visibility: public

7.12.24 TArrayBufIndex.RemoveRecordFromIndex

Synopsis:

Declaration:
```
procedure RemoveRecordFromIndex(const ABookmark: TBufBookmark)
                                      ;  Override
```

Visibility: public

7.12.25 TArrayBufIndex.EndUpdate

Synopsis:

Declaration: `procedure EndUpdate;  Override`

Visibility: public

7.13 TBufBlobStream

7.13.1 Description

`TBufBlobStream` is a `TStream` descendant which implements a Blob Stream for TBufDataset. TBufBlobStream has members which represent the Dataset and Field for the Blob, its Buffer, and modification flags. TBufBlobStream provides overridden methods used to perform Read, Write, and Seek operations using the buffer allocated for a Blob field.

TBufBlobStream instances are created in the TBufDataset.CreateBlobStream method.

See also: TCustomBufDataset.CreateBlobStream (244)

7.13.2 Method overview

Page	Method	Description
223	Create	Constructor for the class instance.
223	Destroy	Destructor for the class instance.

7.13.3 TBufBlobStream.Create

Synopsis: Constructor for the class instance.

Declaration: `constructor Create(Field: TBlobField; Mode: TBlobStreamMode)`

Visibility: public

Description: Constructor for the class instance.

7.13.4 TBufBlobStream.Destroy

Synopsis: Destructor for the class instance.

Declaration: `destructor Destroy;  Override`

Visibility: public

Description: Destroy is the destructor for the class instance.

7.14 TBufDataset

7.14.1 Description

`TBufDataset` is a `TCustomBufDataset` descendant that implements a database-aware buffered dataset. TBufDataset is an in-memory dataset that can be used as a local stand-alone dataset, or it can be used as a local buffer for updates applied to a remote dataset. TBufDataset implements many features similar to those in TClientDataset in Delphi. However, it is **not** meant to be code or function compatible with TClientDataset. It is designed to provide management of the buffers used to access record data, and is used as the ancestor for classes like TSQLQuery and TRESTBufDataset.

As a TDBDataset descendent, it offers access to many of the database features supported in the FCL DB package. As a TDataset descendent, it also offers access to familiar navigation and data handling methods in the ancestor class. TBufDataset has features that allow local storage and retrieval of field definitions and record data. In addition, facilities are provided that allow local indexing of record data

in the dataset. There is a comprehensive parser/expression evaluator available that allows complete support for Filters in the dataset.

TBufDataset sets the visibility for properties and methods defined in the ancestor class.

Additional information about using TBufDataset can be found on the Lazarus Wiki at: How to write in-memory database applications in Lazarus/FPC (TBufDataset).

See also: TCustomBufDataset (237), TDataset (496), TDBDataset (542)

7.14.2 Property overview

Page	Properties	Access	Description
225	Active		
225	AfterCancel		
226	AfterClose		
226	AfterDelete		
226	AfterEdit		
226	AfterInsert		
226	AfterOpen		
226	AfterPost		
226	AfterScroll		
225	AutoCalcFields		
227	BeforeCancel		
227	BeforeClose		
227	BeforeDelete		
227	BeforeEdit		
227	BeforeInsert		
227	BeforeOpen		
227	BeforePost		
228	BeforeScroll		
229	CancelChangesOnRefresh	rw	
224	FieldDefs		
225	Filter		
225	Filtered		
224	MaxIndexesCount		
228	OnCalcFields		
228	OnDeleteError		
228	OnEditError		
228	OnFilterRecord		
228	OnNewRecord		
228	OnPostError		
225	ReadOnly		

7.14.3 TBufDataset.MaxIndexesCount

Declaration: `Property MaxIndexesCount :`

Visibility: published

Access:

7.14.4 TBufDataset.FieldDefs

Declaration: `Property FieldDefs :`

Visibility: published

Access:

7.14.5 TBufDataset.Active

Declaration: `Property Active :`

Visibility: published

Access:

7.14.6 TBufDataset.AutoCalcFields

Declaration: `Property AutoCalcFields :`

Visibility: published

Access:

7.14.7 TBufDataset.Filter

Declaration: `Property Filter :`

Visibility: published

Access:

7.14.8 TBufDataset.Filtered

Declaration: `Property Filtered :`

Visibility: published

Access:

7.14.9 TBufDataset.ReadOnly

Declaration: `Property ReadOnly :`

Visibility: published

Access:

7.14.10 TBufDataset.AfterCancel

Declaration: `Property AfterCancel :`

Visibility: published

Access:

7.14.11 TBufDataset.AfterClose

Declaration: `Property AfterClose :`

Visibility: published

Access:

7.14.12 TBufDataset.AfterDelete

Declaration: `Property AfterDelete :`

Visibility: published

Access:

7.14.13 TBufDataset.AfterEdit

Declaration: `Property AfterEdit :`

Visibility: published

Access:

7.14.14 TBufDataset.AfterInsert

Declaration: `Property AfterInsert :`

Visibility: published

Access:

7.14.15 TBufDataset.AfterOpen

Declaration: `Property AfterOpen :`

Visibility: published

Access:

7.14.16 TBufDataset.AfterPost

Declaration: `Property AfterPost :`

Visibility: published

Access:

7.14.17 TBufDataset.AfterScroll

Declaration: `Property AfterScroll :`

Visibility: published

Access:

7.14.18 TBufDataset.BeforeCancel

Declaration: `Property BeforeCancel :`

Visibility: published

Access:

7.14.19 TBufDataset.BeforeClose

Declaration: `Property BeforeClose :`

Visibility: published

Access:

7.14.20 TBufDataset.BeforeDelete

Declaration: `Property BeforeDelete :`

Visibility: published

Access:

7.14.21 TBufDataset.BeforeEdit

Declaration: `Property BeforeEdit :`

Visibility: published

Access:

7.14.22 TBufDataset.BeforeInsert

Declaration: `Property BeforeInsert :`

Visibility: published

Access:

7.14.23 TBufDataset.BeforeOpen

Declaration: `Property BeforeOpen :`

Visibility: published

Access:

7.14.24 TBufDataset.BeforePost

Declaration: `Property BeforePost :`

Visibility: published

Access:

7.14.25 TBufDataset.BeforeScroll

Declaration: `Property BeforeScroll :`

Visibility: published

Access:

7.14.26 TBufDataset.OnCalcFields

Declaration: `Property OnCalcFields :`

Visibility: published

Access:

7.14.27 TBufDataset.OnDeleteError

Declaration: `Property OnDeleteError :`

Visibility: published

Access:

7.14.28 TBufDataset.OnEditError

Declaration: `Property OnEditError :`

Visibility: published

Access:

7.14.29 TBufDataset.OnFilterRecord

Declaration: `Property OnFilterRecord :`

Visibility: published

Access:

7.14.30 TBufDataset.OnNewRecord

Declaration: `Property OnNewRecord :`

Visibility: published

Access:

7.14.31 TBufDataset.OnPostError

Declaration: `Property OnPostError :`

Visibility: published

Access:

7.14.32 TBufDataset.CancelChangesOnRefresh

Declaration: `Property CancelChangesOnRefresh : Boolean`

Visibility: published

Access: Read,Write

7.15 TBufIndex

7.15.1 Description

`TBufIndex` is a `TObject` descendant which defines the interface used to implement indexes in TBufDataset. TBufIndex provides access to the Dataset with field values for the index, and methods to perform record navigation and index maintenance.

TBufIndex contains many virtual and abstract methods that should be implemented in a descendent class which provides a specific index implementation, such as: TDoubleLinkedBufIndex, TUniDirectionalBufIndex, and TArrayBufIndex.

TBufIndex is the type used for the TBufDataset.BufferIndex property.

See also: TDoubleLinkedBufIndex (256), TUniDirectionalBufIndex (263), TArrayBufIndex (218)

7.15.2 Method overview

Page	Method	Description
235	AddRecord	Adds a record to the index.
234	BeginUpdate	
233	BookmarkValid	Indicates the specified Bookmark is valid.
232	CanScrollForward	Indicates if the index can be scrolled forward.
233	CompareBookmarks	Gets the relative order for the specified Bookmarks.
231	Create	Constructor for the class instance.
233	DoScrollForward	Implements actions need to scroll forward in the index.
235	EndUpdate	
231	GetCurrent	Gets the record buffer for the current record in the dataset.
232	GetRecord	Gets prior/next record relative to the specified bookmark.
233	GotoBookmark	Moves the index position to the specified Bookmark.
234	InitialiseIndex	Initializes the index.
234	InitialiseSpareRecord	Initializes values in the spare record for the index.
235	InsertRecordBeforeCurrentRecord	Inserts a record before the current record in the index order.
235	OrderCurrentRecord	
234	ReleaseSpareRecord	Releases resources allocated to the spare record for the index.
235	RemoveRecordFromIndex	Remove the record at the specified bookmark from the index.
232	RestoreCurrentRecord	Restores the stored record buiffer to the current record.
234	SameBookmarks	Indicates if the specified Bookmarks are for the same record.
231	ScrollBackward	Moves to the prior record in the index.
231	ScrollFirst	Moves to the first record in the index.
231	ScrollForward	Moves to the next record in the index.
231	ScrollLast	Moves to the last record in the index.
232	SetToFirstRecord	Sets the index to the first record in the index order.
232	SetToLastRecord	Sets the index to the last record in the index order.
233	StoreCurrentRecIntoBookmark	
232	StoreCurrentRecord	Stores the record buffer for the current record.
233	StoreSpareRecIntoBookmark	

7.15.3 Property overview

Page	Properties	Access	Description
237	BookmarkSize	r	Size (number of bytes) needed for Bookmarks in the index.
236	CurrentBuffer	r	Current record buffer in the index.
236	CurrentRecord	r	Current record in the index.
236	IsInitialized	r	Indicates if the index has been initialized.
237	RecNo	rw	Active record number in the index.
236	SpareBuffer	r	Spare record buffer for the index.
236	SpareRecord	r	Spare record for the index.

7.15.4 TBufIndex.Create

Synopsis: Constructor for the class instance.

Declaration: `constructor Create(const ADataset: TCustomBufDataset); Virtual`

Visibility: public

Description: `Create` is the constructor for the class instance. Create calls the inherited constructor, and stores the ADataset parameter to the internal member used for the TCustomBufDataset class instance.

7.15.5 TBufIndex.ScrollBackward

Synopsis: Moves to the prior record in the index.

Declaration: `function ScrollBackward : TGetResult; Virtual; Abstract`

Visibility: public

Description: `ScrollBackward` is a TGetResult function used to scrolling to the previous record in the index order. ScrollBackward implements the behavior needed to support the `TDataset.MoveBy` method using the index order for the dataset.

ScrollBackward is an abstract virtual method method, and must be implemented in a descendent class. The return value is a TGetResult enumeration value that indicates the result for the scroll request.

See also: TGetResult (449), TDataset.MoveBy (514)

7.15.6 TBufIndex.ScrollForward

Synopsis: Moves to the next record in the index.

Declaration: `function ScrollForward : TGetResult; Virtual; Abstract`

Visibility: public

7.15.7 TBufIndex.GetCurrent

Synopsis: Gets the record buffer for the current record in the dataset.

Declaration: `function GetCurrent : TGetResult; Virtual; Abstract`

Visibility: public

7.15.8 TBufIndex.ScrollFirst

Synopsis: Moves to the first record in the index.

Declaration: `function ScrollFirst : TGetResult; Virtual; Abstract`

Visibility: public

7.15.9 TBufIndex.ScrollLast

Synopsis: Moves to the last record in the index.

Declaration: `procedure ScrollLast; Virtual; Abstract`

Visibility: public

7.15.10 TBufIndex.GetRecord

Synopsis: Gets prior/next record relative to the specified bookmark.

Declaration:
```
function GetRecord(ABookmark: PBufBookmark; GetMode: TGetMode)
                        : TGetResult;  Virtual
```

Visibility: public

Description: `GetRecord` is used to get the prior/next record relative to the specified bookmark. GetRecord is a virtual method that should be overridden in a descendent class; the implementation in TBufIndex simply returns the value `grError` as the return value for the method.

Please note that GetRecord should **not** change the current record in the dataset on exit.

7.15.11 TBufIndex.SetToFirstRecord

Synopsis: Sets the index to the first record in the index order.

Declaration:
```
procedure SetToFirstRecord;  Virtual;  Abstract
```

Visibility: public

7.15.12 TBufIndex.SetToLastRecord

Synopsis: Sets the index to the last record in the index order.

Declaration:
```
procedure SetToLastRecord;  Virtual;  Abstract
```

Visibility: public

7.15.13 TBufIndex.StoreCurrentRecord

Synopsis: Stores the record buffer for the current record.

Declaration:
```
procedure StoreCurrentRecord;  Virtual;  Abstract
```

Visibility: public

7.15.14 TBufIndex.RestoreCurrentRecord

Synopsis: Restores the stored record buiffer to the current record.

Declaration:
```
procedure RestoreCurrentRecord;  Virtual;  Abstract
```

Visibility: public

7.15.15 TBufIndex.CanScrollForward

Synopsis: Indicates if the index can be scrolled forward.

Declaration:
```
function CanScrollForward : Boolean;  Virtual;  Abstract
```

Visibility: public

7.15.16 TBufIndex.DoScrollForward

Synopsis: Implements actions need to scroll forward in the index.

Declaration:
```
procedure DoScrollForward;  Virtual;  Abstract
```

Visibility: public

7.15.17 TBufIndex.StoreCurrentRecIntoBookmark

Synopsis:

Declaration:
```
procedure StoreCurrentRecIntoBookmark(const ABookmark: PBufBookmark)
                                      ;  Virtual;  Abstract
```

Visibility: public

7.15.18 TBufIndex.StoreSpareRecIntoBookmark

Synopsis:

Declaration:
```
procedure StoreSpareRecIntoBookmark(const ABookmark: PBufBookmark)
                                    ;  Virtual;  Abstract
```

Visibility: public

7.15.19 TBufIndex.GotoBookmark

Synopsis: Moves the index position to the specified Bookmark.

Declaration:
```
procedure GotoBookmark(const ABookmark: PBufBookmark);  Virtual
                       ;  Abstract
```

Visibility: public

7.15.20 TBufIndex.BookmarkValid

Synopsis: Indicates the specified Bookmark is valid.

Declaration:
```
function BookmarkValid(const ABookmark: PBufBookmark) : Boolean
                       ;  Virtual
```

Visibility: public

7.15.21 TBufIndex.CompareBookmarks

Synopsis: Gets the relative order for the specified Bookmarks.

Declaration:
```
function CompareBookmarks(const ABookmark1: PBufBookmark;
                          const ABookmark2: PBufBookmark) : Integer
                          ;  Virtual
```

Visibility: public

7.15.22 TBufIndex.SameBookmarks

Synopsis: Indicates if the specified Bookmarks are for the same record.

Declaration:
```
function SameBookmarks(const ABookmark1: PBufBookmark;
                       const ABookmark2: PBufBookmark) : Boolean
                       ;  Virtual
```

Visibility: public

7.15.23 TBufIndex.InitialiseIndex

Synopsis: Initializes the index.

Declaration:
```
procedure InitialiseIndex;  Virtual;  Abstract
```

Visibility: public

Description: Initializes the index.

7.15.24 TBufIndex.InitialiseSpareRecord

Synopsis: Initializes values in the spare record for the index.

Declaration:
```
procedure InitialiseSpareRecord(const ASpareRecord: TRecordBuffer)
                                ;  Virtual;  Abstract
```

Visibility: public

Description: Initializes values in the spare record for the index.

7.15.25 TBufIndex.ReleaseSpareRecord

Synopsis: Releases resources allocated to the spare record for the index.

Declaration:
```
procedure ReleaseSpareRecord;  Virtual;  Abstract
```

Visibility: public

Description: Releases resources allocated to the spare record for the index.

7.15.26 TBufIndex.BeginUpdate

Synopsis:

Declaration:
```
procedure BeginUpdate;  Virtual;  Abstract
```

Visibility: public

Description:

7.15.27 TBufIndex.AddRecord

Synopsis: Adds a record to the index.

Declaration:
```
procedure AddRecord;  Virtual;  Abstract
```

Visibility: public

Description: Adds a record to the end of the index as the new last record (spare record). AddRecord is used in the GetNextPacket method.

7.15.28 TBufIndex.InsertRecordBeforeCurrentRecord

Synopsis: Inserts a record before the current record in the index order.

Declaration:
```
procedure InsertRecordBeforeCurrentRecord(const ARecord: TRecordBuffer)
                                          ;  Virtual;  Abstract
```

Visibility: public

Description: Inserts a record before the current record using the sort order for the active index.

7.15.29 TBufIndex.RemoveRecordFromIndex

Synopsis: Remove the record at the specified bookmark from the index.

Declaration:
```
procedure RemoveRecordFromIndex(const ABookmark: TBufBookmark);  Virtual
                                ;  Abstract
```

Visibility: public

7.15.30 TBufIndex.OrderCurrentRecord

Synopsis:

Declaration:
```
procedure OrderCurrentRecord;  Virtual;  Abstract
```

Visibility: public

Description:

7.15.31 TBufIndex.EndUpdate

Synopsis:

Declaration:
```
procedure EndUpdate;  Virtual;  Abstract
```

Visibility: public

Description:

7.15.32 TBufIndex.SpareRecord

Synopsis: Spare record for the index.

Declaration: `Property SpareRecord : TRecordBuffer`

Visibility: public

Access: Read

Description: Spare record for the index.

7.15.33 TBufIndex.SpareBuffer

Synopsis: Spare record buffer for the index.

Declaration: `Property SpareBuffer : TRecordBuffer`

Visibility: public

Access: Read

Description: Spare record buffer for the index.

7.15.34 TBufIndex.CurrentRecord

Synopsis: Current record in the index.

Declaration: `Property CurrentRecord : TRecordBuffer`

Visibility: public

Access: Read

Description: Current record in the index.

7.15.35 TBufIndex.CurrentBuffer

Synopsis: Current record buffer in the index.

Declaration: `Property CurrentBuffer : Pointer`

Visibility: public

Access: Read

Description: Current record buffer in the index.

7.15.36 TBufIndex.IsInitialized

Synopsis: Indicates if the index has been initialized.

Declaration: `Property IsInitialized : Boolean`

Visibility: public

Access: Read

Description: Indicates if the index has been initialized.

7.15.37 TBufIndex.BookmarkSize

Synopsis: Size (number of bytes) needed for Bookmarks in the index.

Declaration: `Property BookmarkSize : Integer`

Visibility: public

Access: Read

Description: Size (number of bytes) needed for Bookmarks in the index.

7.15.38 TBufIndex.RecNo

Synopsis: Active record number in the index.

Declaration: `Property RecNo : LongInt`

Visibility: public

Access: Read,Write

Description: Active record number in the index.

7.16 TCustomBufDataset

7.16.1 Description

`TCustomBufDataset` is a `TDBDataset` descendant that implements the ancestor class for a database-aware buffered dataset. TCustomBufDataset is an in-memory dataset that can be used as a local stand-alone dataset, or it can be used as a local buffer for updates applied to a remote dataset. TCustomBufDataset implements many features similar to those in TClientDataset in Delphi. However, it is **not** meant to be code or function compatible with TClientDataset. It is designed to provide management of the buffers used to access record data, and is used as the ancestor for classes like `TSQLQuery` and `TRESTBufDataset`.

As a TDBDataset descendant, it offers access to many of the database features supported in the FCL DB package. As a TDataset descendant, it also offers access to familiar navigation and data handling methods in the ancestor class. TCustomBufDataset has features that allow local storage and retrieval of field definitions and record data. In addition, facilities are provided that allow local indexing of record data in the dataset. There is a comprehensive parser/expression evaluator available that allows complete support for Filters in the dataset.

Do not create instances of TCustomBufDataset. Use the TBufDataset descendant instead.

Additional information about using TBufDataset can be found on the Lazarus Wiki at: How to write in-memory database applications in Lazarus/FPC (TBufDataset).

See also: TBufDataset (223), TDBDataset (542), TDataset (496)

7.16.2 Method overview

Page	Method	Description
244	AddIndex	Adds an index definition to the dataset.
240	ApplyUpdates	Applies pending updates to the dataset.
249	BookmarkValid	Determines if the specified Bookmark is valid for the dataset.
241	CancelUpdates	Cancels pending updates in the dataset.
248	Clear	Clears the content in the dataset.
245	ClearIndexes	Clears index storage in the dataset.
249	CompareBookmarks	Gets the relative sort order for the specified Bookmarks.
249	CopyFromDataset	Loads field definitions and optional data from another dataset.
239	Create	Constructor for the class instance.
244	CreateBlobStream	Creates a Blob stream for the specified field with the given permissions.
248	CreateDataset	Creates the dataset using its field definitions or bound fields.
241	Destroy	Destructor for the class instance.
245	GetDatasetPacket	Builds a data packet representing the content in the buffered dataset.
240	GetFieldData	
247	LoadFromFile	Loads the dataset from the specified file using the given format.
246	LoadFromStream	Loads the dataset from the specified stream using the given data format.
242	Locate	Locates the first record with fields having the specified values.
242	Lookup	Gets values from the first record with fields that match the search condition.
241	MergeChangeLog	Frees update buffers and Blob update buffer allocated for the dataset.
241	RevertRecord	Reverts the current record to its original (un-edited) values.
247	SaveToFile	Saves the dataset to the specified file using the given data format.
246	SaveToStream	Saves the dataset to the specified stream using the given data format.
245	SetDatasetPacket	Retrieves and applies the data packet with the content for the dataset.
240	SetFieldData	
243	UpdateStatus	Gets the update status for the current record in the dataset.

7.16.3 Property overview

Page	Properties	Access	Description
250	ChangeCount	r	Number of pending changes for the dataset.
251	FileName	rw	File name on the local file system used to load or store the dataset.
253	IndexDefs	r	Index definitions for the dataset.
253	IndexFieldNames	rw	Field names included in the custom index.
253	IndexName	rw	Name of the selected index for the dataset.
251	ManualMergeChangeLog	rw	Indicates if the update change log can be manually merged.
250	MaxIndexesCount	rw	Maximum number of indexes available in the dataset.
252	OnUpdateError	rw	Event handler signalled when an error occurs while updating records.
252	PacketRecords	rw	Number of records allowed in a data packet handled by the packet reader.
251	ReadOnly	rw	Indicates if records can be added, deleted, or modified in the dataset.
254	UniDirectional	rw	Indicates if the dataset is for uni-directional navigation only.

7.16.4 TCustomBufDataset.Create

Synopsis: Constructor for the class instance.

Declaration: `constructor Create(AOwner: TComponent);  Override`

Visibility: public

Description: `Create` is an overridden constructor for the class instance. Create calls the inherited constructor using the value in AOwner as the owner of the class instance. Create sets the default value for internal members used in the implementation of the buffered dataset, including:

- Parser for data packets (`Nil`)
- Manual MergeChangeLog handling (**False**)
- Default AutoInc field value (**-1**)
- Update Buffer count (**0**)
- Blob Buffer count (**0**)
- Blob Update Buffer count (**0**)

Creates sets the value for public and published properties, including:

- MaxIndexesCount (**2**)
- PacketRecords (**10**)

Create allocates resources needed for the BufIndexdefs and BufIndexes properties.

See also: TCustomBufDataset.MaxIndexesCount (250), TCustomBufDataset.PacketRecords (252)

7.16.5 TCustomBufDataset.GetFieldData

Synopsis:

Declaration:
```
function GetFieldData(Field: TField; Buffer: Pointer;
                      NativeFormat: Boolean) : Boolean;  Override
function GetFieldData(Field: TField; Buffer: Pointer) : Boolean
                      ;  Override
```

Visibility: public

Description:

7.16.6 TCustomBufDataset.SetFieldData

Synopsis:

Declaration:
```
procedure SetFieldData(Field: TField; Buffer: Pointer;
                       NativeFormat: Boolean);  Override
procedure SetFieldData(Field: TField; Buffer: Pointer);  Override
```

Visibility: public

Description:

7.16.7 TCustomBufDataset.ApplyUpdates

Synopsis: Applies pending updates to the dataset.

Declaration:
```
procedure ApplyUpdates;  Virtual;  Overload
procedure ApplyUpdates(MaxErrors: Integer);  Virtual;  Overload
```

Visibility: public

Description: `ApplyUpdates` is an overloaded procedure used to apply pending updates for the dataset.

ApplyUpdates calls CheckBrowseMode and gets a bookmark for the current record in the dataset. The bookmark is used to return the dataset its original record after updates are applied and the dataset has been refreshed.

MaxErrors is the threshold where errors encountered in the process cause it to be aborted. The value `0` (`zero`) indicates no errors are allowed during the apply updates process.

ApplyUpdates uses the internal update buffers allocated for the dataset to perform the actions required in the method. The ApplyRecUpdate method is called for each of the update buffers. An exception raised in ApplyRecUpdate is handled in the method. When the value in MaxErrors is exceeded, the process is aborted. Otherwise, the update buffer is skipped. If the OnUpdateError exception handler has been assigned, it is signalled using the required arguments and its Response value is captured in the method. If the process is to be aborted, an exception is raised to indicate the condition.

When an update buffer is successfully applied, resources allocated to its OldValuesBuffer, Bookmark data, and the update buffer itself are freed. If all pending updates are applied successfully, and ManualMergeChangeLog contains `False`, the MergeChangeLog method is called.

ApplyUpdates calls Resync to fresh the records displayed in the dataset prior to exiting from the method.

See also: TDataset.CheckBrowseMode (504), TRecUpdateBuffer (218), TCustomBufDataset.OnUpdateError (252), TCustomBufDataset.ManualMergeChangeLog (251), TCustomBufDataset.MergeChangeLog (241), TDataset.Resync (516)

7.16.8 TCustomBufDataset.MergeChangeLog

Synopsis: Frees update buffers and Blob update buffer allocated for the dataset.

Declaration: `procedure MergeChangeLog`

Visibility: public

Description: `MergeChangeLog` is a procedure used to free update buffers and Blob update buffer allocated for the dataset. MergeChangeLog is called from the ApplyUpdates method.

See also: TCustomBufDataset.ApplyUpdates (240)

7.16.9 TCustomBufDataset.RevertRecord

Synopsis: Reverts the current record to its original (un-edited) values.

Declaration: `procedure RevertRecord`

Visibility: public

Description: `RevertRecord` is a procedure used to revert changes in the current record to their original (un-edited) values. RevertRecord calls the CheckBrowseMode method to ensure that the dataset is Active, and to perform event notifications for a change in dataset state.

RevertRecord checks for an update buffer in the current record, and when found calls the CancelRecordUpdateBuffer method and removes the update buffer. The Resync method is called to refresh the records in the dataset.

See also: TDataset.CheckBrowseMode (504), TDataset.Active (526), TCustomBufDataset.Resync (237)

7.16.10 TCustomBufDataset.CancelUpdates

Synopsis: Cancels pending updates in the dataset.

Declaration: `procedure CancelUpdates; Virtual`

Visibility: public

Description: `CancelUpdates` is a procedure used to cancel pending updates to records in the dataset. CancelUpdates calls the CheckBrowseMode method to ensure that the dataset is Active, and to perform event notifications for a change in dataset state.

CancelUpdates checks for existing update buffers allocated in the dataset. When update buffers exist, the CancelRecordUpdateBuffer method is called for the bookmark in each of the update buffers.

CancelUpdates restores the record position when able, and calls Resync to refresh the records available in the dataset.

See also: TDataset.CheckBrowseMode (504), TDataset.Active (526), TCustomBufDataset.Resync (237)

7.16.11 TCustomBufDataset.Destroy

Synopsis: Destructor for the class instance.

Declaration: `destructor Destroy; Override`

Visibility: public

Description: `Destroy` is the overridden destructor for the class instance. Destroy calls the Close method if the dataset contains `True` in the Active property.

Destroy frees resources allocated to the internal buffers for records, and Blob fields. The ClearIndexes method is called to free resources allocated for index storage mechanisms. Destroy calls the inherited destructor.

7.16.12 TCustomBufDataset.Locate

Synopsis: Locates the first record with fields having the specified values.

Declaration:
```
function Locate(const KeyFields: string; const KeyValues: Variant;
                Options: TLocateOptions) : Boolean;  Override
```

Visibility: public

Description: `Locate` is an overridden `Boolean` function used to locate the first record with fields that match the specified search values. Locate implements the method defined in the ancestor class.

KeyFields is a delimited list of field names searched in the method. The delimiter character is '**;**' (**Semicolon**).

KeyValues is a Variant type with the field values required for the specified KeyFields. When KeyFields contains a single field name, KeyValues can be represented using the same data type as the field value. When KeyFields contains more than one field name, it is a Variant array.

Options is a TLocateOptions parameter that indicates the locate options enabled in the method. Options can contain zero (**0**) or more values from the TLocateOption enumeration.

Locate calls the inherited method to ensure that the dataset supports bi-directional navigation. An Exception is raised if the UniDirectional property contains `True`. The value in the Active property is checked to ensure that the dataset has been opened. An Exception is raised if Active contains `False`. No actions are performed in the method when IsEmpty returns `True`.

Locate initializes an internal TDBCompareStruct instance that is used when searching field values using the SearchFields and Options arguments. The search is performed by using the Filter feature; the filter fields are set to the values in KeyValues. Each record in the dataset is visited and the OnFilterRecord event is signalled to determine if the record meets the Filter criteria. The search ends when a record is found that matches the search values.

The record position is changed to the marching record, or positioned at the end-of-file when no matching record is found.

The return value is `True` if a record was located that matched the search values.

See also: TDataset.Locate (513), TCustomBufDataset.UniDirectional (254), TDataset.Active (526), TDataset.Filter (525), TDataset.OnFilterRecord (525), TLocateOptions (450), TDBCompareStruct (214)

7.16.13 TCustomBufDataset.Lookup

Synopsis: Gets values from the first record with fields that match the search condition.

Declaration:
```
function Lookup(const KeyFields: string; const KeyValues: Variant;
                const ResultFields: string) : Variant;  Override
```

Visibility: public

Description: `Lookup` is an overridden `Variant` function used to get values from the first record with fields that match the specified search values. Lookup re-implements the method defined in the ancestor class. Lookup allows searching one or more fields for corresponding values, and returning a set of field values from the matching record.

KeyFields is a delimited list of field names searched in the method. The delimiter character is '**;**' (**Semicolon**). The following example would represent the fields **LASTNAME** and **FIRSTNAME**:

```
'LASTNAME; FIRSTNAME'
```

KeyValues is a Variant type with the field values required for the specified KeyFields. When KeyFields contains a single field name, KeyValues can be represented using the same data type as the field value. When KeyFields contains more than one field name, it is a Variant array. Using the previous values for KeyFields, the following would represent the values for '**JOHN SMITH**':

```
VarArrayOf([ 'SMITH', 'JOHN' ])
```

ResultFields contains one or more field names which determine the field values to include in the return value for the method. Like KeyFields, it is a delimited list of field names using '**;**' (**Semicolon**) as the delimiter character.

The return value is a Variant type that contains a value for each of the field names specified in the ResultFields argument. When a matching record is located, the return value is a Variant array. The return value is `Null` if a record is not found matching the specified KeyValues in its KeyFields.

```
var
  AVarValues: Variant;
  ABufDataset: TBufDataset;

// ...

AVarValues := ABufDataset.Lookup('lastname;firstname',
  VarArrayOf(['SMITH', 'JOHN']), 'lastname;firstname;birthdate;emailaddress');

if not VarIsNull(AVarValues) then
  DoSomethingWith(AVarValues)
else
  WriteLn('No records match the search criteria');
```

Lookup does not change the record position in the dataset. A temporary TBookmark is used to record the current record in the dataset. The record position is restored prior to exiting from the method.

Lookup calls the Locate method to search records in the dataset using the KeyFields and KeyValues arguments. The Locate method may raise an Exception if the dataset is configured for UniDirectional access or when it is not Active.

See also: TCustomBufDataset.Locate (242), TDataset.Active (526), TCustomBufDataset.UniDirectional (254)

7.16.14 TCustomBufDataset.UpdateStatus

Synopsis: Gets the update status for the current record in the dataset.

Declaration: `function UpdateStatus : TUpdateStatus;  Override`

Visibility: public

Description: `UpdateStatus` is an overridden `TUpdateStatus` function used to get the update status for the current record in the dataset. UpdateStatus re-implements the method defined in the ancestor class. The return value is a member of the TUpdateStatus enumeration, and includes the following values:

usUnmodified Record has not been modified

usModified Record has been modified

usInserted Records was appended or inserted

usDeleted Record has been deleted

UpdateStatus calls GetActiveRecordUpdateBuffer to get the update buffer for the ActiveRecord. When an update buffer exists for the record, its UpdateKind property is used as the return value for the method. The return value is `usUnmodified` when an update buffer does not exist for ActiveRecord.

See also: TUpdateStatus (455), TDataset.ActiveRecord (496)

7.16.15 TCustomBufDataset.CreateBlobStream

Synopsis: Creates a Blob stream for the specified field with the given permissions.

Declaration:
```
function CreateBlobStream(Field: TField; Mode: TBlobStreamMode)
                                 : TStream;  Override
```

Visibility: public

Description: Creates a Blob stream for the specified field with the given permissions.

See also: TDataset.CreateBlobStream (506), TField (549), TBlobStreamMode (441)

7.16.16 TCustomBufDataset.AddIndex

Synopsis: Adds an index definition to the dataset.

Declaration:
```
procedure AddIndex(const AName: string; const AFields: string;
                        AOptions: TIndexOptions; const ADescFields: string;
                        const ACaseInsFields: string);  Virtual
```

Visibility: public

Description: `AddIndex` is a procedure used to add an index definition to the dataset. AddIndex calls CheckBiDirectional to ensure that the UniDirectional property contains `False`. An exception is raised if UniDirectional contains `True`.

AddIndex checks for a valid value in the AFields argument. An Exception is raised when AFields is an empty string (''). In addition, an index added to an Active dataset cannot cause the index count to exceed the value in MaxIndexesCount. An Exception is raised if the index count would be larger than the value in MaxIndexesCount.

AddIndex creates a TBufDatasetIndex instance by calling InternalAddIndex using the arguments passed to the method. The new index is temporary when the Active property contains `True`; it will be discarded when the dataset is closed.

See also: TCustomBufDataset.UniDirectional (254), TDataset.Active (526), TBufDatasetIndex (213)

7.16.17 TCustomBufDataset.ClearIndexes

Synopsis: Clears index storage in the dataset.

Declaration: `procedure ClearIndexes`

Visibility: public

Description: `ClearIndexes` is a procedure procedure used to clear the storage for all indexes in the dataset. ClearIndexes uses the index definitions in the BufIndexDefs property and calls the ClearIndex method for each of the indexes.

ClearIndexes calls CheckInactive to ensure that the dataset has its Active property set to `False`. Indexes cannot be cleared if the dataset has been opened.

ClearIndexes is used when the value in the UniDirectional property is changed to `False`, and in the destructor for the class instance.

See also: TDataset.CheckInactive (496), TCustomBufDataset.UniDirectional (254)

7.16.18 TCustomBufDataset.SetDatasetPacket

Synopsis: Retrieves and applies the data packet with the content for the dataset.

Declaration: `procedure SetDatasetPacket(AReader: TDataPacketHandler)`

Visibility: public

Description: SetDatasetPacket is a procedure used to retrieve and apply the data packet with the content for the dataset.

AReader is a TDataPacketReader instance that is used to read and apply the content for the buffered dataset. It is temporarily assigned to an internal member in the calls instance.

SetDatasetPacket calls the Open method to load the field definitions and record data for the dataset.

SetDatasetPacket is used in the implementation of the LoadFromStream method.

See also: TDataset.Open (515), TCustomBufDataset.LoadFromStream (246), TDataPacketReader (214)

7.16.19 TCustomBufDataset.GetDatasetPacket

Synopsis: Builds a data packet representing the content in the buffered dataset.

Declaration: `procedure GetDatasetPacket(AWriter: TDataPacketHandler)`

Visibility: public

Description: `GetDatasetPacket` is a procedure used to build the data packet that represents the content in the buffered dataset. GetDatasetPacket ensures that record buffers and update buffers in the dataset are normalized and applied using the data packet handler (n.b. I won't call it a reader if it writes) in AWriter.

GetDatasetPacket sequentially processes records using the currently selected index for the dataset. Update buffers for a given record are consolidate in the record buffer, and the record buffer to stored to the data packet handler. The current record position is not changed when the process has been completed.

GetDatasetPacket stores the current value for an auto-increment field used in the dataset in the AWriter argument. The internal data packet handler for the dataset is set to `Nil` when the method is completed.

GetDatasetPacket is used in the implementation of the SaveToStream method.

See also: TDataPacketReader (214), TCustomBufDataset.SaveToStream (246)

7.16.20 TCustomBufDataset.LoadFromStream

Synopsis: Loads the dataset from the specified stream using the given data format.

Declaration: `procedure LoadFromStream(AStream: TStream; Format: TDataPacketFormat)`

Visibility: public

Description: `LoadFromStream` is a procedure used to load field definitions and record data for the dataset from the specified stream. LoadFromStream is an overloaded variant of the method defined in the ancestor class and includes a parameter for the desired data packet format.

AStream is a TStream descendant with the values to load in the dataset. Format indicates the expected format for data packets in the stream. It is a variable argument that can be modified when a packet reader is located for the stream.

LoadFromStream calls CheckBiDirectional to ensure that the UniDirectional property in the dataset is set to `False`. An exception is raised when UniDirectional contains `True`. Index creation requires navigating in a non-default record order that would not be supported in a uni-directional dataset.

LoadFromStream creates a temporary TDataPacketReader class instance using the value specified in the Format argument. Format is a variable argument and can be modified when SetDatasetPacket is used to examine the packet reader and its stream. Calling SetDatasetPacket sets in motion a relatively complicated sequence of events that eventually get the field definitions and record data loaded into the dataset. The journey starts with Open.

LoadFromStream is used in the implementation of the LoadFromFile method.

See also: TCustomBufDataset.UniDirectional (254), TCustomBufDataset.SetDatasetPacket (245), TCustomBufDataset.LoadFromFile (247), TDataset.Open (515), TDataPacketFormat (214)

7.16.21 TCustomBufDataset.SaveToStream

Synopsis: Saves the dataset to the specified stream using the given data format.

Declaration: `procedure SaveToStream(AStream: TStream; Format: TDataPacketFormat)`

Visibility: public

Description: SaveToStream is a procedure used to store the content in the buffered dataset to a stream using the specified data format.

AStream is a TStream decendant where the field definitions and record data in the dataset will be stored.

Format is a TDataPacketFormat enumeration value that indicates the format used to stored dataset values into the stream. The default value is dfBinary, and causes the TFpcBinaryDatapacketReader to be used to write the content for the dataset. Use another value from the TDataPacketFormat enumeration to select a different registered data packet handler that handles the associated data format.

SaveToStream calls the CheckBiDirectional method to ensure that the dataset is opened for bi-directional record navigation. An Exception is raised if the UniDirectional property contains True. A temporary TDataPacketReader instance is created using the specified Format. SaveToStream calls the GetDatasetPacket method using the data packet handler to store the content in the buffered dataset to the stream.

SaveToStream is used in the implementation of the SaveToFile method.

See also: TCustomBufDataset.UniDirectional (254), TCustomBufDataset.SaveToFile (247), TDataPacketFormat (214), RegisterDatapacketReader (216), TDataPacketReaderClass (214), TDataPacketReader (214)

7.16.22 TCustomBufDataset.LoadFromFile

Synopsis: Loads the dataset from the specified file using the given format.

Declaration:
```
procedure LoadFromFile(const AFileName: string;
                       Format: TDataPacketFormat)
```

Visibility: public

Description: `LoadFromFile` is a procedure used to load field definitions and record data from a file stored using a given data format.

AFileName is a String with the name of the file on the local file system. AFileName can contain optional path information, and should include the base name and extension for the file. For example:

```
ABufDataset.LoadFromFile('/usr/data/sample.bds');
```

or

```
ABufDataset.LoadFromFile('c:\usr\data\sample.bds');
```

The default value for AFileName is an empty string (''). When AFileName is an empty string, the value in the FileName property is used to load the content for the dataset. An Exception will be raised if both AFileName and FileName contain an empty String (''), or when the file name does not exist on the local file system.

Format is a TDataPacketFormat enumeration value that indicates the storage format used for the content in the file. The default value for Format is dfAny, and indicates that any registered data packet handler can be used to read the values in the file. Other values in the TDataPacketFormat enumeration indicate a specific registered data packet handler supporting a specific format should be used.

LoadFromFile creates a temporary TFileStream instance for the specified file name, and calls the LoadFromStream method to load the dataset from the stream using the specified file Format.

See also: TCustomBufDataset.FileName (251), TCustomBufDataset.LoadFromStream (246)

7.16.23 TCustomBufDataset.SaveToFile

Synopsis: Saves the dataset to the specified file using the given data format.

Declaration:
```
procedure SaveToFile(const AFileName: string; Format: TDataPacketFormat)
```

Visibility: public

Description: `SaveToFile` is a procedure used to save the content from the buffered dataset to the specified file name using a given data format.

AFileName is a String with the name of the file on the local file system. AFileName can contain optional path information, and should include the base name and extension for the file. For example:

```
ABufDataset.SaveToFile('/usr/data/sample.bds');
```

or

```
ABufDataset.SaveToFile('c:\usr\data\sample.bds');
```

The default value for AFileName is an empty string (''). When AFileName is omitted, the value in the FileName property will be used to save the content for the dataset. An Exception will be raised if both AFileName and FileName contain an empty String (''').

Format is a TDataPacketFormat enumeration value that indicates the storage format to use when writing the content for the file. The default value for Format is dfBinary, and indicates that the TFpcBinaryDatapacketReader should be used to write the values in the file. Other values in the TDataPacketFormat enumeration indicate a specific registered data packet handler supporting a specific format should be used.

SaveToFile creates a temporary TFileStream instance for the specified file name, and calls the SaveToStream method to store the dataset to the stream in the required Format.

See also: TCustomBufDataset.FileName (251), TCustomBufDataset.SaveToStream (246)

7.16.24 TCustomBufDataset.CreateDataset

Synopsis: Creates the dataset using its field definitions or bound fields.

Declaration: `procedure CreateDataset`

Visibility: public

Description: `CreateDataset` is a procedure used to create the structure for a buffered dataset using the field definitions or bound fields defined in the class instance. CreateDataset calls CheckInactive to ensure that the dataset has not been opened. An exception is raised if the Active property contains `True`.

CreateDataset uses the FieldDefs and Fields properties to determine which property contains the structure for the dataset.

Preference is given to the FieldDefs property; it will be used even when bound fields have been created in the Fields property. The CreateFields method is called to created a bound field for each of the items in the FieldDefs collection.

The Fields property is used when there are no field definitions present in the FieldDefs property. The InitFieldDefsFromFields method is called to create a field definition in FieldDefs for each of the items in Fields collection. The BindFields method is called to link the items in Fields to the corresponding field definition in FieldDefs.

An exception is raised if both FieldDefs and Fields are empty.

CreateDataset sets the default value for an auto-increment field in the dataset to `1`. CreateDataset also temporarily clears any value assigned to the FileName property; this is done to ensure that fields and record data in an existing file are not loaded when the dataset is Opened. The method is designed to create a dataset with field definitions but no record data. The value in FileName is restored after Open has been called in the method.

See also: TDataset.Active (526), TDataset.FieldDefs (521), TDataset.Fields (524), TDataset.Open (515), TCustomBufDataset.FileName (251)

7.16.25 TCustomBufDataset.Clear

Synopsis: Clears the content in the dataset.

Declaration: `procedure Clear`

Visibility: public

Description: `Clear` is a procedure used to remove the content in the buffered dataset. Clear calls the Close method to ensure that the dataset has set its Active property to `False` and removed indexes and their storage. Clear removes field definition and bound fields in the FieldDefs and Fields properties.

See also: TDataset.Close (505), TDataset.FieldDefs (521), TDataset.Fields (524)

7.16.26 TCustomBufDataset.BookmarkValid

Synopsis: Determines if the specified Bookmark is valid for the dataset.

Declaration:
```
function BookmarkValid(ABookmark: TBookMark) : Boolean;  Override
```

Visibility: public

Description: `BookmarkValid` is an overridden `Boolean` function used to determine if the Bookmark specified in ABookmark is valid for the currently selected index in the dataset. BookmarkValid re-implements the method defined in the ancestor class.

BookmarkValid requires an index be selected and assigned to the CurrentIndexBuf property. The return value is `False` when no index is available to validate the bookmark. The return value is `True` when ABookmark represents a valid bookmark in the selected index.

See also: TBufIndex.BookmarkValid (233)

7.16.27 TCustomBufDataset.CompareBookmarks

Synopsis: Gets the relative sort order for the specified Bookmarks.

Declaration:
```
function CompareBookmarks(Bookmark1: TBookMark; Bookmark2: TBookMark)
                                  : LongInt;  Override
```

Visibility: public

Description: CompareBookmarks is an overridden LongInt function used to get the relative sort order for the specified Bookmarks. CompareBookmarks re-implements the method defined in the ancestor class.

The return value indicates the relative sort order for the compared bookmark values, and uses the following values and meanings:

0 Compared bookmarks have the same value

1 Bookmark1 is not assigned, or Bookmark1 occurs after Bookmark2 in the index

-1 Bookmark2 is not assigned, or the current index has not been assigned (default value), or Bookmark1 occurs before Bookmark2 in the index

When the CurentIndexBuf property is assigned, its CompareBookmarks method is used to compare the bookmarks and get the return value for the method.

See also: TBufIndex (229)

7.16.28 TCustomBufDataset.CopyFromDataset

Synopsis: Loads field definitions and optional data from another dataset.

Declaration:
```
procedure CopyFromDataset(DataSet: TDataSet; CopyData: Boolean)
```

Visibility: public

Description: `CopyFromDataset` is a procedure used to copy field definitions and optional record data for the dataset specified in the Dataset argument. CopyData indicates if record data is included in the copy operation. When CopyData contains `False`, only the field definitions in DataSet are copied to the buffered dataset.

CopyFromDataset calls the Close method prior to performing actions that clear field definitions and bound fields in the buffered dataset. A field definition is added to FieldDefs for each of the fields in DataSet. CreateDataset is called to bind Fields to the new field definitions, and to initialize index storage for the buffered dataset.

When CopyData contains `True`, record data from DataSet is copied to the class instance on a row-by-row and field-by-field basis. Append is called to add a new record. Internal lists with field definitions and bound fields are used to copy field values in the record (including Blob fields). Null field values are not copied in the method. The DataType for the field is used to read/write the field values. For blob fields, a temporary stream is used to read/write the field value. The Post method is called to store appended values in the buffered dataset. If an exception is raised, the Cancel method is called to clear the update and the exception is re-raised.

The record position in DataSet is restored to its original position prior to exiting from the method.

See also: TDataset.Close (505), TDataset.Cancel (504), TDataset.FieldDefs (521)

7.16.29 TCustomBufDataset.ChangeCount

Synopsis: Number of pending changes for the dataset.

Declaration: `Property ChangeCount : Integer`

Visibility: public

Access: Read

Description: `ChangeCount` is a read-only `Integer` property that indicates the number of pending changes in the update buffers allocated for the dataset. Update buffers are maintained in an internal TRecordsUpdateBuffer member when methods that add, delete, or modify record data are called.

ChangeCount can be used in an application to determine if ApplyUpdates, CancelUpdates, or RevertRecord should be called prior to closing the dataset.

See also: TCustomBufDataset.ApplyUpdates (240), TCustomBufDataset.CancelUpdates (241), TCustomBufDataset.RevertRecord (241)

7.16.30 TCustomBufDataset.MaxIndexesCount

Synopsis: Maximum number of indexes available in the dataset.

Declaration: `Property MaxIndexesCount : Integer`

Visibility: public

Access: Read,Write

Description: `MaxIndexesCount` is an `Integer` property which indicates the maximum number of indexes available in the buffered dataset. The default value for the property is `2`; representing the automatically created default index (`'DEFAULT_ORDER'`) and custom index (`'CUSTOM_ORDER'`).

The value in MaxIndexesCount is assigned when the dataset is created, and updated when an index is added using the AddIndex method. Setting the value in the MaxIndexesCount property requires the dataset to be inactive (Active property contains `False`). An exception is raised if the dataset is

Active, or the new value for the property is less than 2. You should not need to directly assign the value for the MaxIndexesCount property.

MaxIndexesCount is used in the BufferOffset method to determine the amount of space reserved for TBufRecLinkItem items in a record buffer for the dataset.

See also: TDataset.Active (526), TCustomBufDataset.AddIndex (244)

7.16.31 TCustomBufDataset.ReadOnly

Synopsis: Indicates if records can be added, deleted, or modified in the dataset.

Declaration: `Property ReadOnly : Boolean`

Visibility: public

Access: Read,Write

Description: `ReadOnly` is a `Boolean` property which indicates if records can be added, deleted, or modified in the buffered dataset. The default value for the property is `False`.

ReadOnly is used, along with UniDirectional, when getting the value for the CanModify property. The dataset can be modified when both source properties contain the value `False`, which results in setting CanModify to `True`. Applications can set the value in ReadOnly to `False` to ensure that the dataset is not changeable regardless of the value in the UniDirectional property.

Use the items in the FieldDefs property to control whether individual fields defined for the dataset include the read-only field attribute. Use the items in the Fields property to determine if a field bound to a field definition has its ReadOnly property set.

See also: TCustomBufDataset.UniDirectional (254), TDataset.CanModify (519), TDataset.FieldDefs (521), TDataset.Fields (524)

7.16.32 TCustomBufDataset.ManualMergeChangeLog

Synopsis: Indicates if the update change log can be manually merged.

Declaration: `Property ManualMergeChangeLog : Boolean`

Visibility: public

Access: Read,Write

Description: `ManualMergeChangeLog` is a `Boolean` property which indicates if the update change log for the dataset can be manually merged. The default value for the property is `False`.

7.16.33 TCustomBufDataset.FileName

Synopsis: File name on the local file system used to load or store the dataset.

Declaration: `Property FileName : TFileName`

Visibility: published

Access: Read,Write

Description: `FileName` is a `TFileName` property that contains a file name on the local file system used to load and/or store the content for the buffered dataset. FileName can contain optional path information needed to access the file, and must contain a valid file name and extension for the local file system. For example:

```
ABufDataset.FileName := '/usr/data/sample.bds';
```

or

```
ABufDataset.FileName := 'c:\usr\data\sample.bds';
```

The value in FileName is used in methods which load and/or save field definitions and record data for the dataset, such as LoadFromFile and SaveToFile. In these methods, FileName is used as the default value for an omitted file name argument in the method(s). FileName is used in the implementation of other methods such as: InternalInitFieldDef, IntLoadFieldDefsFromFile, and InternalOpen.

FileName is also used in the DoBeforeClose method called when the value in the Active property is changed from `True` to `False`. In this method, FileName is passed to SaveToFile as an argument prior to exiting from the method.

See also: TCustomBufDataset.LoadFromFile (247), TCustomBufDataset.SaveToFile (247), TDataset.Open (515), TDataset.Close (505)

7.16.34 TCustomBufDataset.PacketRecords

Synopsis: Number of records allowed in a data packet handled by the packet reader.

Declaration: `Property PacketRecords : Integer`

Visibility: published

Access: Read,Write

Description: `PacketRecords` is an `Integer` property that indicates the number of records allowed in a data packet handled by the packet reader. The default value for the property is `10`, and is intended to minimize memory and network overhead when processing data packets for the dataset. The value in PacketRecords is used in the FetchAll and GetNextPacket methods.

Additional validation is performed when setting the value for PacketRecords to ensure that the new property value is not `-1`. An exception is raised when `-1` is the value for the property.

7.16.35 TCustomBufDataset.OnUpdateError

Synopsis: Event handler signalled when an error occurs while updating records.

Declaration: `Property OnUpdateError : TResolverErrorEvent`

Visibility: published

Access: Read,Write

Description: `OnUpdateError` is a `TResolverErrorEvent` property that represents the event handler signalled when an error occurs while applying updates to records in the buffered dataset. OnUpdateError allows an application to perform actions needed when a database exception occurs in the `ApplyUpdates` method.

Applications can assign a procedure to the event handler that implements the TResolverErrorEvent signature to respond to the event notification. The procedure must set the value in its Response argument to indicate whether the condition is handled, ignored, or can be re-raised in the calling method.

See also: TResolverErrorEvent (215), TCustomBufDataset.ApplyUpdates (240)

7.16.36 TCustomBufDataset.IndexDefs

Synopsis: Index definitions for the dataset.

Declaration: `Property IndexDefs : TIndexDefs`

Visibility: published

Access: Read

Description: `IndexDefs` is a read-only `TIndexDefs` property that contains the index definitions for the buffered dataset. Read access to the property is redirected to an internal TBufDatasetIndexDefs member used for the BufIndexDefs and BufIndexes properties.

See also: TBufDatasetIndexDefs (237), TIndexDefs (599)

7.16.37 TCustomBufDataset.IndexName

Synopsis: Name of the selected index for the dataset.

Declaration: `Property IndexName : string`

Visibility: published

Access: Read,Write

Description: `IndexName` is a `String` property that contains the name for the selected index for the buffered dataset. The value in IndexName is read from the corresponding property in CurrentIndexBuf (when assigned).

If an empty string (**''**) is assigned to the property, the the default index is selected for the dataset and the property is updated to `'DEFAULT_ORDER'`. Any other value assigned to the property is compared to the index definitions for the dataset. If an index cannot be located with the specified name, a Database exception is raised. The index with the specified name (and its storage mechanism) are stored in the CurrentIndexDef and CurrentIndexBuf properties. If the dataset is Active, the Resync method is called to enable the new record order for the dataset.

See also: TDataset.Active (526), TDataset.Resync (516)

7.16.38 TCustomBufDataset.IndexFieldNames

Synopsis: Field names included in the custom index.

Declaration: `Property IndexFieldNames : string`

Visibility: published

Access: Read,Write

Description: `IndexFieldNames` is a `String` property which contains a delimited list of field names used to construct the custom index for the dataset. Field names are separated by a **';'** (Semicolon) delimiter. An optional directive can be included after the field name to indicate that the field should be in descending sort order in the index. For example:

```
ABufDataset.IndexFieldNames := 'LASTNAME; FIRSTNAME; UPDATETS DESC';
```

Please note that the leading Space character before the **DESC** directive is **required**.

When reading the value in IndexFieldNames, the current index is examined to determine if any of the field names in the property also appear in the descending fields for the index. The `' DESC'` directive is added to the field name to indicate the sort order used in the index.

Setting the value for the property to an empty string ('') causes the default index (`'DEFAULT_ORDER'`) to be used as the selected index in the CurrentIndexDef property. When the new property value is not an empty string, and the dataset is Active, the BuildCustomIndex method is called to populate index storage with values for the specified field names.

See also: TCustomBufDataset.CurrentIndexDef (237), TDataset.Active (526)

7.16.39 TCustomBufDataset.UniDirectional

Synopsis: Indicates if the dataset is for uni-directional navigation only.

Declaration: `Property UniDirectional : Boolean`

Visibility: published

Access: Read,Write

Description: `UniDirectional` is a `Boolean` property which indicates if the dataset is limited to forward navigation through its records. The default value for the property is `False`.

UniDirectional is used, along with ReadOnly, to determine if the dataset can be modified. When either property contains `True`, the dataset cannot be changed.

UniDirectional also affects the index storage metchanisms created when indexes are initialized. When UniDIrectional is `True`, the TUniDirectionalBufIndex type (which omits bookmarks and record numbers) is used for index storage mechanisms. In addition, the custom index (`'CUSTOM_ORDER'`) is skipped for the uni-directional dataset.

Setting the value in UniDirectional requires the dataset to be inactive (Active contains `False`) to allow existing indexes to be cleared and rebuilt. Use the Close method to cl;osr the dataset prior to setting the value in the UniDIrectional property.

See also: TDataset.ReadOnly (496), TDataset.Active (526)

7.17 TDataPacketHandler

7.17.1 Method overview

Page	Method	Description
255	Create	
256	FinalizeStoreRecords	
255	GetCurrentRecord	
255	GetRecordRowState	
255	GotoNextRecord	
255	InitLoadRecords	
255	LoadFieldDefs	
256	RecognizeStream	
255	RestoreRecord	
255	StoreFieldDefs	
256	StoreRecord	

7.17.2 TDataPacketHandler.Create

Declaration: `constructor Create(ADataSet: TCustomBufDataset; AStream: TStream) ; Virtual`

Visibility: public

7.17.3 TDataPacketHandler.LoadFieldDefs

Declaration: `procedure LoadFieldDefs(var AnAutoIncValue: Integer); Virtual ; Abstract`

Visibility: public

7.17.4 TDataPacketHandler.InitLoadRecords

Declaration: `procedure InitLoadRecords; Virtual; Abstract`

Visibility: public

7.17.5 TDataPacketHandler.GetCurrentRecord

Declaration: `function GetCurrentRecord : Boolean; Virtual; Abstract`

Visibility: public

7.17.6 TDataPacketHandler.GetRecordRowState

Declaration: `function GetRecordRowState(out AUpdOrder: Integer) : TRowState; Virtual ; Abstract`

Visibility: public

7.17.7 TDataPacketHandler.RestoreRecord

Declaration: `procedure RestoreRecord; Virtual; Abstract`

Visibility: public

7.17.8 TDataPacketHandler.GotoNextRecord

Declaration: `procedure GotoNextRecord; Virtual; Abstract`

Visibility: public

7.17.9 TDataPacketHandler.StoreFieldDefs

Declaration: `procedure StoreFieldDefs(AnAutoIncValue: Integer); Virtual; Abstract`

Visibility: public

7.17.10 TDataPacketHandler.StoreRecord

Declaration: `procedure StoreRecord(ARowState: TRowState; AUpdOrder: Integer) ; Virtual; Abstract`

Visibility: public

7.17.11 TDataPacketHandler.FinalizeStoreRecords

Declaration: `procedure FinalizeStoreRecords; Virtual; Abstract`

Visibility: public

7.17.12 TDataPacketHandler.RecognizeStream

Declaration: `class function RecognizeStream(AStream: TStream) : Boolean; Virtual ; Abstract`

Visibility: public

7.18 TDoubleLinkedBufIndex

7.18.1 Description

`TDoubleLinkedBufIndex` is a `TBufIndex` descendant that implements an index using a doubly-linked list. Nodes in the doubly-linked list are implemented using the TBufRecLinkItem record type and the PBufRecLinkItem pointer type. TDoubleLinkedBufIndex provides two sentinel nodes that represent the first and last items in the linked list.

TDoubleLinkedBufIndex is the type used to implement the default index (`'DEFAULT_ORDER'`) in TBufDataset.

See also: TBufIndex (229), TBufRecLinkItem (217), PBufRecLinkItem (214), TBufDataset (223)

7.18.2 Method overview

Page	Method	Description
261	AddRecord	Adds a record to the index.
260	BeginUpdate	
259	CanScrollForward	Indicates if the index can be scrolled towards the end of the index.
260	CompareBookmarks	
259	DoScrollForward	Implements actions needed to scroll forward using the index.
261	EndUpdate	
257	GetCurrent	
258	GetRecord	
259	GotoBookmark	Navigates the index to the specified Bookmark.
260	InitialiseIndex	Initializes the index.
260	InitialiseSpareRecord	Initializes the spare record in the index.
261	InsertRecordBeforeCurrentRecord	Inserts a record prior to the current record in the index.
261	OrderCurrentRecord	
260	ReleaseSpareRecord	Releases the spare record in the index.
261	RemoveRecordFromIndex	Removes the record at the specified Bookmark from the index.
259	RestoreCurrentRecord	
260	SameBookmarks	Compares Bookmark values for ordering in the index.
257	ScrollBackward	Scrolls the index toward the beginning of the index.
258	ScrollFirst	Scrolls to the first entry in the index.
257	ScrollForward	Scrolls the index towards the end of the index.
258	ScrollLast	Scrolls to the last entry in the index.
258	SetToFirstRecord	
258	SetToLastRecord	
259	StoreCurrentRecIntoBookmark	
258	StoreCurrentRecord	
259	StoreSpareRecIntoBookmark	

7.18.3 TDoubleLinkedBufIndex.ScrollBackward

Synopsis: Scrolls the index toward the beginning of the index.

Declaration: `function ScrollBackward : TGetResult;  Override`

Visibility: public

7.18.4 TDoubleLinkedBufIndex.ScrollForward

Synopsis: Scrolls the index towards the end of the index.

Declaration: `function ScrollForward : TGetResult;  Override`

Visibility: public

7.18.5 TDoubleLinkedBufIndex.GetCurrent

Synopsis:

Declaration: `function GetCurrent : TGetResult;  Override`

Visibility: public

7.18.6 TDoubleLinkedBufIndex.ScrollFirst

Synopsis: Scrolls to the first entry in the index.

Declaration: `function ScrollFirst : TGetResult;  Override`

Visibility: public

7.18.7 TDoubleLinkedBufIndex.ScrollLast

Synopsis: Scrolls to the last entry in the index.

Declaration: `procedure ScrollLast;  Override`

Visibility: public

7.18.8 TDoubleLinkedBufIndex.GetRecord

Synopsis:

Declaration:
```
function GetRecord(ABookmark: PBufBookmark; GetMode: TGetMode)
                         : TGetResult;  Override
```

Visibility: public

7.18.9 TDoubleLinkedBufIndex.SetToFirstRecord

Synopsis:

Declaration: `procedure SetToFirstRecord;  Override`

Visibility: public

7.18.10 TDoubleLinkedBufIndex.SetToLastRecord

Synopsis:

Declaration: `procedure SetToLastRecord;  Override`

Visibility: public

7.18.11 TDoubleLinkedBufIndex.StoreCurrentRecord

Synopsis:

Declaration: `procedure StoreCurrentRecord;  Override`

Visibility: public

7.18.12 TDoubleLinkedBufIndex.RestoreCurrentRecord

Synopsis:

Declaration: `procedure RestoreCurrentRecord;  Override`

Visibility: public

7.18.13 TDoubleLinkedBufIndex.CanScrollForward

Synopsis: Indicates if the index can be scrolled towards the end of the index.

Declaration: `function CanScrollForward : Boolean;  Override`

Visibility: public

7.18.14 TDoubleLinkedBufIndex.DoScrollForward

Synopsis: Implements actions needed to scroll forward using the index.

Declaration: `procedure DoScrollForward;  Override`

Visibility: public

7.18.15 TDoubleLinkedBufIndex.StoreCurrentRecIntoBookmark

Synopsis:

Declaration:
```
procedure StoreCurrentRecIntoBookmark(const ABookmark: PBufBookmark)
                                      ;  Override
```

Visibility: public

7.18.16 TDoubleLinkedBufIndex.StoreSpareRecIntoBookmark

Synopsis:

Declaration:
```
procedure StoreSpareRecIntoBookmark(const ABookmark: PBufBookmark)
                                    ;  Override
```

Visibility: public

7.18.17 TDoubleLinkedBufIndex.GotoBookmark

Synopsis: Navigates the index to the specified Bookmark.

Declaration: `procedure GotoBookmark(const ABookmark: PBufBookmark);  Override`

Visibility: public

7.18.18 TDoubleLinkedBufIndex.CompareBookmarks

Synopsis:

Declaration:
```
function CompareBookmarks(const ABookmark1: PBufBookmark;
                          const ABookmark2: PBufBookmark) : Integer
                          ;  Override
```

Visibility: public

7.18.19 TDoubleLinkedBufIndex.SameBookmarks

Synopsis: Compares Bookmark values for ordering in the index.

Declaration:
```
function SameBookmarks(const ABookmark1: PBufBookmark;
                       const ABookmark2: PBufBookmark) : Boolean
                       ;  Override
```

Visibility: public

7.18.20 TDoubleLinkedBufIndex.InitialiseIndex

Synopsis: Initializes the index.

Declaration:
```
procedure InitialiseIndex;  Override
```

Visibility: public

7.18.21 TDoubleLinkedBufIndex.InitialiseSpareRecord

Synopsis: Initializes the spare record in the index.

Declaration:
```
procedure InitialiseSpareRecord(const ASpareRecord: TRecordBuffer)
                                ;  Override
```

Visibility: public

7.18.22 TDoubleLinkedBufIndex.ReleaseSpareRecord

Synopsis: Releases the spare record in the index.

Declaration:
```
procedure ReleaseSpareRecord;  Override
```

Visibility: public

7.18.23 TDoubleLinkedBufIndex.BeginUpdate

Synopsis:

Declaration:
```
procedure BeginUpdate;  Override
```

Visibility: public

7.18.24 TDoubleLinkedBufIndex.AddRecord

Synopsis: Adds a record to the index.

Declaration:
```
procedure AddRecord;  Override
```

Visibility: public

7.18.25 TDoubleLinkedBufIndex.InsertRecordBeforeCurrentRecord

Synopsis: Inserts a record prior to the current record in the index.

Declaration:
```
procedure InsertRecordBeforeCurrentRecord(const ARecord: TRecordBuffer)
                                                ;  Override
```

Visibility: public

7.18.26 TDoubleLinkedBufIndex.RemoveRecordFromIndex

Synopsis: Removes the record at the specified Bookmark from the index.

Declaration:
```
procedure RemoveRecordFromIndex(const ABookmark: TBufBookmark)
                                      ;  Override
```

Visibility: public

7.18.27 TDoubleLinkedBufIndex.OrderCurrentRecord

Synopsis:

Declaration:
```
procedure OrderCurrentRecord;  Override
```

Visibility: public

7.18.28 TDoubleLinkedBufIndex.EndUpdate

Synopsis:

Declaration:
```
procedure EndUpdate;  Override
```

Visibility: public

7.19 TFpcBinaryDatapacketHandler

7.19.1 Method overview

Page	Method	Description
262	Create	
263	FinalizeStoreRecords	
262	GetCurrentRecord	
262	GetRecordRowState	
263	GotoNextRecord	
262	InitLoadRecords	
262	LoadFieldDefs	
263	RecognizeStream	
263	RestoreRecord	
262	StoreFieldDefs	
263	StoreRecord	

7.19.2 TFpcBinaryDatapacketHandler.Create

Declaration: `constructor Create(ADataSet: TCustomBufDataset; AStream: TStream) ; Override`

Visibility: public

7.19.3 TFpcBinaryDatapacketHandler.LoadFieldDefs

Declaration: `procedure LoadFieldDefs(var AnAutoIncValue: Integer); Override`

Visibility: public

7.19.4 TFpcBinaryDatapacketHandler.StoreFieldDefs

Declaration: `procedure StoreFieldDefs(AnAutoIncValue: Integer); Override`

Visibility: public

7.19.5 TFpcBinaryDatapacketHandler.InitLoadRecords

Declaration: `procedure InitLoadRecords; Override`

Visibility: public

7.19.6 TFpcBinaryDatapacketHandler.GetCurrentRecord

Declaration: `function GetCurrentRecord : Boolean; Override`

Visibility: public

7.19.7 TFpcBinaryDatapacketHandler.GetRecordRowState

Declaration: `function GetRecordRowState(out AUpdOrder: Integer) : TRowState ; Override`

Visibility: public

7.19.8 TFpcBinaryDatapacketHandler.RestoreRecord

Declaration: `procedure RestoreRecord;  Override`

Visibility: public

7.19.9 TFpcBinaryDatapacketHandler.GotoNextRecord

Declaration: `procedure GotoNextRecord;  Override`

Visibility: public

7.19.10 TFpcBinaryDatapacketHandler.StoreRecord

Declaration:
```
procedure StoreRecord(ARowState: TRowState; AUpdOrder: Integer)
                      ;  Override
```

Visibility: public

7.19.11 TFpcBinaryDatapacketHandler.FinalizeStoreRecords

Declaration: `procedure FinalizeStoreRecords;  Override`

Visibility: public

7.19.12 TFpcBinaryDatapacketHandler.RecognizeStream

Declaration: `class function RecognizeStream(AStream: TStream) : Boolean;  Override`

Visibility: public

7.20 TUniDirectionalBufIndex

7.20.1 Description

`TUniDirectionalBufIndex` is a `TBufIndex` descendant that implements a uni-directional index. TUniDirectionalBufIndex does not require bookmarks used for navigation; it is uni-directional and the next record is always available in the buffers allocated for the index.

See also: TBufIndex (229)

7.20.2 Method overview

Page	Method	Description
267	AddRecord	
267	BeginUpdate	
265	CanScrollForward	
266	DoScrollForward	
267	EndUpdate	
264	GetCurrent	
266	GotoBookmark	
266	InitialiseIndex	
266	InitialiseSpareRecord	
267	InsertRecordBeforeCurrentRecord	
267	OrderCurrentRecord	
266	ReleaseSpareRecord	
267	RemoveRecordFromIndex	
265	RestoreCurrentRecord	
264	ScrollBackward	
264	ScrollFirst	
264	ScrollForward	
265	ScrollLast	
265	SetToFirstRecord	
265	SetToLastRecord	
266	StoreCurrentRecIntoBookmark	
265	StoreCurrentRecord	
266	StoreSpareRecIntoBookmark	

7.20.3 TUniDirectionalBufIndex.ScrollBackward

Synopsis:

Declaration: `function ScrollBackward : TGetResult;  Override`

Visibility: public

7.20.4 TUniDirectionalBufIndex.ScrollForward

Synopsis:

Declaration: `function ScrollForward : TGetResult;  Override`

Visibility: public

7.20.5 TUniDirectionalBufIndex.GetCurrent

Synopsis:

Declaration: `function GetCurrent : TGetResult;  Override`

Visibility: public

7.20.6 TUniDirectionalBufIndex.ScrollFirst

Synopsis:

Declaration: `function ScrollFirst : TGetResult;  Override`

Visibility: public

7.20.7 TUniDirectionalBufIndex.ScrollLast

Synopsis:

Declaration: `procedure ScrollLast;  Override`

Visibility: public

7.20.8 TUniDirectionalBufIndex.SetToFirstRecord

Synopsis:

Declaration: `procedure SetToFirstRecord;  Override`

Visibility: public

7.20.9 TUniDirectionalBufIndex.SetToLastRecord

Synopsis:

Declaration: `procedure SetToLastRecord;  Override`

Visibility: public

7.20.10 TUniDirectionalBufIndex.StoreCurrentRecord

Synopsis:

Declaration: `procedure StoreCurrentRecord;  Override`

Visibility: public

7.20.11 TUniDirectionalBufIndex.RestoreCurrentRecord

Synopsis:

Declaration: `procedure RestoreCurrentRecord;  Override`

Visibility: public

7.20.12 TUniDirectionalBufIndex.CanScrollForward

Synopsis:

Declaration: `function CanScrollForward : Boolean;  Override`

Visibility: public

7.20.13 TUniDirectionalBufIndex.DoScrollForward

Synopsis:

Declaration:
```
procedure DoScrollForward;  Override
```

Visibility: public

7.20.14 TUniDirectionalBufIndex.StoreCurrentRecIntoBookmark

Synopsis:

Declaration:
```
procedure StoreCurrentRecIntoBookmark(const ABookmark: PBufBookmark)
                                      ;  Override
```

Visibility: public

7.20.15 TUniDirectionalBufIndex.StoreSpareRecIntoBookmark

Synopsis:

Declaration:
```
procedure StoreSpareRecIntoBookmark(const ABookmark: PBufBookmark)
                                    ;  Override
```

Visibility: public

7.20.16 TUniDirectionalBufIndex.GotoBookmark

Synopsis:

Declaration:
```
procedure GotoBookmark(const ABookmark: PBufBookmark);  Override
```

Visibility: public

7.20.17 TUniDirectionalBufIndex.InitialiseIndex

Synopsis:

Declaration:
```
procedure InitialiseIndex;  Override
```

Visibility: public

7.20.18 TUniDirectionalBufIndex.InitialiseSpareRecord

Synopsis:

Declaration:
```
procedure InitialiseSpareRecord(const ASpareRecord: TRecordBuffer)
                                ;  Override
```

Visibility: public

7.20.19 TUniDirectionalBufIndex.ReleaseSpareRecord

Synopsis:

Declaration:
```
procedure ReleaseSpareRecord;  Override
```

Visibility: public

7.20.20 TUniDirectionalBufIndex.BeginUpdate

Synopsis:

Declaration: `procedure BeginUpdate;  Override`

Visibility: public

7.20.21 TUniDirectionalBufIndex.AddRecord

Synopsis:

Declaration: `procedure AddRecord;  Override`

Visibility: public

7.20.22 TUniDirectionalBufIndex.InsertRecordBeforeCurrentRecord

Synopsis:

Declaration:
```
procedure InsertRecordBeforeCurrentRecord(const ARecord: TRecordBuffer)
                                          ;  Override
```

Visibility: public

7.20.23 TUniDirectionalBufIndex.RemoveRecordFromIndex

Synopsis:

Declaration:
```
procedure RemoveRecordFromIndex(const ABookmark: TBufBookmark)
                                ;  Override
```

Visibility: public

7.20.24 TUniDirectionalBufIndex.OrderCurrentRecord

Synopsis:

Declaration: `procedure OrderCurrentRecord;  Override`

Visibility: public

7.20.25 TUniDirectionalBufIndex.EndUpdate

Synopsis:

Declaration: `procedure EndUpdate;  Override`

Visibility: public

Chapter 8

Reference for unit 'bufstream'

8.1 Used units

Table 8.1: Used units by unit 'bufstream'

Name	Page
Classes	??
System	??
sysutils	??

8.2 Overview

BufStream implements buffered streams. The streams store all data from (or for) the source stream in a memory buffer, and only flush the buffer when it's full (or refill it when it's empty).

Buffered streams can help in speeding up read or write operations, especially when a lot of small read/write operations are done. They avoid doing a lot of operating system calls.

TReadBufStream (274) is used for reading only, and allows the buffer size to be specified at the time of creation.

TWriteBufStream (275) is used for writing only, and allows the buffer size to be specified at the time of creation.

TBufferedFileStream (269) can be used for reading and writing depending on the file mode specified at the time of creation. By default, it uses an internal buffer with 8 pages using a 4,096 byte page size. Both page count and page size are configurable using methods in the class.

8.3 Constants, types and variables

8.3.1 Constants

```
DefaultBufferCapacity : Integer = 16
```

If no buffer size is specified when the stream is created, then this size is used.

8.4 TBufferedFileStream

8.4.1 Description

`TBufferedFileStream` is a `TFileStream` descendant which implements a buffered file stream. It provides a buffer with multiple pages used for random read / write access in the file stream.

By default, It uses a fixed-size buffer consisting of 8 pages with a 4,096 bytes per page. Both page count and page size configurable using the InitializeCache (272) method in the class. The buffer is automatically maintained when the stream size or position is changed, and when reading or writing content to/from the stream.

Pages which have been modified in the buffer are written to the file stream as needed (all pages are used and a read/write operation is performed), when the Flush method is called, and when the class instance is freed. Thus, the class will never use more than the total size obtained by multiplying the `CacheBlockSize` and `aCacheBlockCount` arguments to InitializeCache (272).

This class is suitable when you need to do a lot of random access to a file: hence the use of different pages of configurable size. If you simply wish to speed up sequential reads you better use TReadBufStream (274) or TWriteBufStream (275) for speeding up sequential writes.

See also: TFileStream (**??**), THandleStream (**??**), TStream (**??**), TReadBufStream (274), TWriteBufStream (275)

8.4.2 Method overview

Page	Method	Description
269	Create	Constructor for the class instance.
270	Destroy	Destructor for the class instance.
272	Flush	Flushes modified pages in the buffer to the file stream.
272	InitializeCache	Re-initializes the internal buffer for the buffered file stream.
271	Read	Reads the specified number of bytes into the Buffer parameter.
270	Seek	Moves the position in the buffer relative to the specified origin.
271	Write	Writes the specified number of bytes in Buffer to the internal page buffer(s).

8.4.3 TBufferedFileStream.Create

Synopsis: Constructor for the class instance.

Declaration:
```
constructor Create(const AFileName: string; Mode: Word)
constructor Create(const AFileName: string; Mode: Word;
                   Rights: Cardinal)
```

Visibility: public

Description: `Create` is the constructor for the class instance. Overloaded variants are provided to match the constructors used in the ancestor class (`TFileStream`).

Create ensures that resources are allocated for the internal buffer. By default, the buffer reserves 8 blocks (pages) with 4,096 bytes per block (page). Create calls `InitializeCache` to allocate resources needed for the internal buffer.

Create calls the inherited constructor using the parameter values passed to the method.

`AFileName` is the qualified path to the file where the content in the stream is stored.

`Mode` contains the file mode used for the file handle in the ancestor class. It uses the following file mode constant values:

fmCreate Creates the file it does not already exist.

fmOpenRead Opens the file for read-only access.

fmOpenWrite Opens the file for write-only access.

fmOpenReadWrite Opens the file for read / write access.

The file mode constants (**except for fmCreate**) can be **OR**'d with sharing mode constants, including:

fmShareCompat Opens the file in DOS-compatibility sharing mode.

fmShareExclusive Locks the file for exclusive use.

fmShareDenyWrite Locks the file and denies write access to other processes.

fmShareDenyRead Locks the file and denies read access to other processes.

fmShareDenyNone Does not lock the file.

`Rights` contains the value used as the file mode on UNIX-like file systems. It contains a value representing the read, write, execute, sticky-bit, setgid, and setuid flags used on the platform. It is ignored for all other platforms, and is significant only when using `fmCreate` in `Mode`.

The `Size` for the internal buffer is updated to use the length of the file stream.

See also: TBufferedFileStream.InitializeCache (272), TBufferedFileStream.Size (269), TFileStream.Create (**??**), TFileStream.Size (**??**)

8.4.4 TBufferedFileStream.Destroy

Synopsis: Destructor for the class instance.

Declaration:
```
destructor Destroy;  Override
```

Visibility: public

Description: `Destroy` is the overridden destructor for the class instance. Destroy ensures that memory allocated to pages in the internal buffer is freed, and that buffer pages are released. Destroy calls the inherited destructor prior to exit.

See also: TFileStream.Destroy (**??**)

8.4.5 TBufferedFileStream.Seek

Synopsis: Moves the position in the buffer relative to the specified origin.

Declaration:
```
function Seek(Offset: LongInt; Origin: Word) : LongInt;  Override
                ;  Overload
function Seek(const Offset: Int64; Origin: TSeekOrigin) : Int64
                ;  Override;  Overload
```

Visibility: public

Description: `Seek` is a method used to change the current position in the buffered file stream by the number of bytes in `Offset` relative to the given `Origin`. Overloaded variants are provided which use `LongInt` or `Int64` types for the Offset parameter, and `Word` or `TSeekOrigin` types for the `Origin` parameter.

Seek is overridden to use the size and position in the internal buffer when positioning the buffered file stream.

The return value contains the actual number of bytes the position was moved relative to the Origin. As with `TStream`, the return value may contain -1 if the stream position was not moved.

See also: THandleStream.Seek (**??**), TStream.Seek (**??**)

8.4.6 TBufferedFileStream.Read

Synopsis: Reads the specified number of bytes into the Buffer parameter.

Declaration: `function Read(var Buffer; Count: LongInt) : LongInt;  Override`

Visibility: public

Description: `Read` is used to read the specified number of bytes in `Count`, and store the values in the `Buffer` parameter. Read is overridden to use the internal buffer for the operation instead directly accessing of the underlying file stream. It locates the page in the buffer with the content for the stream position.

Read maintains the pages in the buffer as needed for the request. This includes writing and recycling older buffer pages, locating the position in the file stream for a new buffer page, and loading the content for a buffer page from the file stream.

`Buffer` is updated with the values copied from the internal buffer. The return value contains the actual number of bytes read from the internal buffer, or 0 when no content is available in the buffer at the current stream position.

Use `Seek` or `Position` to set the stream position (when needed) prior to calling Read.

Errors: Read raises an `EStreamError` exception with the message in `SErrCacheUnexpectedPageDiscard` when a page has been unexpectedly discarded in the buffer.

See also: THandleStream.Read (**??**), THandleStream.Seek (**??**), TStream.Read (**??**), TStream.Seek (**??**), TStream.Position (**??**), EStreamError (**??**)

8.4.7 TBufferedFileStream.Write

Synopsis: Writes the specified number of bytes in Buffer to the internal page buffer(s).

Declaration: `function Write(const Buffer; Count: LongInt) : LongInt;  Override`

Visibility: public

Description: `Write` is a `LongInt` function used to write byte values in `Buffer` to the current position in the buffered file stream. `Count` contains the number of bytes requested in the write operation.

Write is overridden to use the internal buffer in the operation instead of the underlying file stream. It locates the page in the buffer with the content for the buffered stream position.

Write maintains the pages in the buffer as needed for the request. This includes recycling older buffer pages, reading values from the stream for a new buffer page, and storing the new content in the internal buffer.

`Buffer` contains the values stored in the internal buffer in the request. The return value contains the actual number of bytes written, or 0 if the write could not be performed.

Use `Seek` or `Position` to set the buffer position (when needed) prior to calling Write.

Errors: Write raises an `EStreamError` exception with the message in `SErrCacheUnexpectedPageDiscard` when a page has been unexpectedly discarded in the buffer.

See also: TBufferedFileStream.Seek (270), TBufferedFileStream.GetPosition (269), TBufferedFileStream.SetPosition (269), THandleStream.Write (**??**), TStream.Write (**??**), TStream.Position (**??**), EStreamError (**??**)

8.4.8 TBufferedFileStream.Flush

Synopsis: Flushes modified pages in the buffer to the file stream.

Declaration: `procedure Flush`

Visibility: public

Description: `Flush` is used to store modified pages in the internal buffer to the file stream. Flush examines the pages in the buffer to determine if any have been modified using `Write`.

When a "dirty" page is found, the inherited `Seek` method is called to position the stream to the location for the modified page. The inherited `Write` method is called to store content in the modified page buffer to the stream, and the modified flag for the buffer page is reset.

Buffer pages which not been modified are not (re-)written to the file stream.

Errors: Flush raises an `EStreamError` exception with the message in SErrCacheUnableToWriteExpected when the number of bytes written for a page does not match the allocated size for the page.

See also: TBufferedFileStream.Write (271), THandleStream.Seek (**??**), THandleStream.Write (**??**), TStream.Seek (**??**), TStream.Write (**??**), EStreamError (**??**)

8.4.9 TBufferedFileStream.InitializeCache

Synopsis: Re-initializes the internal buffer for the buffered file stream.

Declaration:
```
procedure InitializeCache(const aCacheBlockSize: Integer;
                          const aCacheBlockCount: Integer)
```

Visibility: public

Description: Re-initializes the internal buffer to use the number of blocks (pages) in `aCacheBlockCount` where each block (page) has the size in `aCacheBlockSize`.

`InitializeCache` checks pages in the internal buffer to see if any have been modified, and writes them to the file stream when needed. Memory allocated to an existing buffer page is freed, and the page is discarded.

Values in aCacheBlockCount and aCacheBlockSize are stored internally, and the buffer size is updated to use the size from the file stream.

InitializeCache re-allocates and zero-fills memory used for each of the pages in the buffer prior to exiting from the method.

InitializeCache is called from the `Create` method to allocate buffer pages using the default count and size for the class.

Use `Flush` to write modified values in page buffers to the file stream without re-initializing the internal buffer.

See also: TBufferedFileStream.Flush (272), TBufferedFileStream.Create (269)

8.5 TBufStream

8.5.1 Description

`TBufStream` is the common ancestor for the TReadBufStream (274) and TWriteBufStream (275) streams. It completely handles the buffer memory management and position management. An instance of `TBufStream` should never be created directly. It also keeps the instance of the source stream.

See also: TReadBufStream (274), TWriteBufStream (275)

8.5.2 Method overview

Page	Method	Description
273	Create	Create a new TBufStream instance.
273	Destroy	Destroys the TBufStream instance.

8.5.3 Property overview

Page	Properties	Access	Description
273	Buffer	r	The current buffer.
274	BufferPos	r	Current buffer position.
274	BufferSize	r	Amount of data in the buffer.
274	Capacity	rw	Current buffer capacity.

8.5.4 TBufStream.Create

Synopsis: Create a new TBufStream instance.

Declaration:
```
constructor Create(ASource: TStream; ACapacity: Integer)
constructor Create(ASource: TStream)
```

Visibility: public

Description: Create creates a new TBufStream instance. A buffer of size ACapacity is allocated, and the ASource source (or destination) stream is stored. If no capacity is specified, then DefaultBufferCapacity (268) is used as the capacity.

An instance of TBufStream should never be instantiated directly. Instead, an instance of TReadBufStream (274) or TWriteBufStream (275) should be created.

Errors: If not enough memory is available for the buffer, then an exception may be raised.

See also: TBufStream.Destroy (273), TReadBufStream (274), TWriteBufStream (275)

8.5.5 TBufStream.Destroy

Synopsis: Destroys the TBufStream instance.

Declaration:
```
destructor Destroy;  Override
```

Visibility: public

Description: Destroy destroys the instance of TBufStream. It flushes the buffer, deallocates it, and then destroys the TBufStream instance.

See also: TBufStream.Create (273), TReadBufStream (274), TWriteBufStream (275)

8.5.6 TBufStream.Buffer

Synopsis: The current buffer.

Declaration:
```
Property Buffer : Pointer
```

Visibility: public

Access: Read

Description: Buffer is a pointer to the actual buffer in use.

See also: TBufStream.Create (273), TBufStream.Capacity (274), TBufStream.BufferSize (274)

8.5.7 TBufStream.Capacity

Synopsis: Current buffer capacity.

Declaration: `Property Capacity : Integer`

Visibility: public

Access: Read,Write

Description: `Capacity` is the amount of memory the buffer occupies. To change the buffer size, the capacity can be set. Note that the capacity cannot be set to a value that is less than the current buffer size, i.e. the current amount of data in the buffer.

See also: TBufStream.Create (273), TBufStream.Buffer (273), TBufStream.BufferSize (274), TBufStream.BufferPos (274)

8.5.8 TBufStream.BufferPos

Synopsis: Current buffer position.

Declaration: `Property BufferPos : Integer`

Visibility: public

Access: Read

Description: `BufPos` is the current stream position in the buffer. Depending on whether the stream is used for reading or writing, data will be read from this position, or will be written at this position in the buffer.

See also: TBufStream.Create (273), TBufStream.Buffer (273), TBufStream.BufferSize (274), TBufStream.Capacity (274)

8.5.9 TBufStream.BufferSize

Synopsis: Amount of data in the buffer.

Declaration: `Property BufferSize : Integer`

Visibility: public

Access: Read

Description: `BufferSize` is the actual amount of data in the buffer. This is always less than or equal to the Capacity (274).

See also: TBufStream.Create (273), TBufStream.Buffer (273), TBufStream.BufferPos (274), TBufStream.Capacity (274)

8.6 TReadBufStream

8.6.1 Description

`TReadBufStream` is a read-only buffered stream. It implements the needed methods to read data from the buffer and fill the buffer with additional data when needed.

The stream provides limited forward-seek possibilities.

See also: TBufStream (272), TWriteBufStream (275)

8.6.2 Method overview

Page	Method	Description
275	Read	Reads data from the stream.
275	Seek	Set location in the buffer.

8.6.3 TReadBufStream.Seek

Synopsis: Set location in the buffer.

Declaration:
```
function Seek(const Offset: Int64; Origin: TSeekOrigin) : Int64
                ;  Override
```

Visibility: public

Description: `Seek` sets the location in the buffer. Currently, only a forward seek is allowed. It is emulated by reading and discarding data. For an explanation of the parameters, see TStream.Seek" (**??**).

The seek method needs enhancement to enable it to do a full-featured seek. This may be implemented in a future release of Free Pascal.

Errors: In case an illegal seek operation is attempted, an exception is raised.

See also: TWriteBufStream.Seek (276), TReadBufStream.Read (275)

8.6.4 TReadBufStream.Read

Synopsis: Reads data from the stream.

Declaration:
```
function Read(var ABuffer; ACount: LongInt) : Integer;  Override
```

Visibility: public

Description: `Read` reads at most `ACount` bytes from the stream and places them in `Buffer`. The number of actually read bytes is returned.

`TReadBufStream` first reads whatever data is still available in the buffer, and then refills the buffer, after which it continues to read data from the buffer. This is repeated until `ACount` bytes are read, or no more data is available.

See also: TReadBufStream.Seek (275), TReadBufStream.Read (275)

8.7 TWriteBufStream

8.7.1 Description

`TWriteBufStream` is a write-only buffered stream. It implements the needed methods to write data to the buffer and flush the buffer (i.e., write its contents to the source stream) when needed.

See also: TBufStream (272), TReadBufStream (274)

8.7.2 Method overview

Page	Method	Description
276	Destroy	Remove the `TWriteBufStream` instance from memory.
276	Seek	Set stream position.
276	Write	Write data to the stream.

8.7.3 TWriteBufStream.Destroy

Synopsis: Remove the `TWriteBufStream` instance from memory.

Declaration:
```
destructor Destroy;  Override
```

Visibility: public

Description: `Destroy` flushes the buffer and then calls the inherited Destroy (273).

Errors: If an error occurs during flushing of the buffer, an exception may be raised.

See also: Create (273), TBufStream.Destroy (273)

8.7.4 TWriteBufStream.Seek

Synopsis: Set stream position.

Declaration:
```
function Seek(const Offset: Int64; Origin: TSeekOrigin) : Int64
                  ;  Override
```

Visibility: public

Description: `Seek` always raises an `EStreamError` exception, except when the seek operation would not alter the current position.

A later implementation may perform a proper seek operation by flushing the buffer and doing a seek on the source stream.

See also: TWriteBufStream.Write (276), TReadBufStream.Seek (275)

8.7.5 TWriteBufStream.Write

Synopsis: Write data to the stream.

Declaration:
```
function Write(const ABuffer; ACount: LongInt) : Integer;  Override
```

Visibility: public

Description: `Write` writes at most `ACount` bytes from `ABuffer` to the stream. The data is written to the internal buffer first. As soon as the internal buffer is full, it is flushed to the destination stream, and the internal buffer is filled again. This process continues till all data is written (or an error occurs).

Errors: An exception may occur if the destination stream has problems writing.

See also: TWriteBufStream.Seek (276)

Chapter 9

Reference for unit 'CacheCls'

9.1 Used units

Table 9.1: Used units by unit 'CacheCls'

Name	Page
System	??
sysutils	??

9.2 Overview

The CacheCls unit implements a caching class: similar to a hash class, it can be used to cache data, associated with string values (keys). The class is calls `TCache`

9.3 Constants, types and variables

9.3.1 Resource strings

```
SInvalidIndex = 'Invalid index %i'
```

Message shown when an invalid index is passed.

9.3.2 Types

```
PCacheSlot = ^TCacheSlot
```

Pointer to TCacheSlot (278) record.

```
PCacheSlotArray = ^TCacheSlotArray
```

Pointer to TCacheSlotArray (278) array.

```
TCacheSlotArray = Array[0..MaxIntdivSizeOf(TCacheSlot)-1] of
  TCacheSlot
```

TCacheSlotArray is an array of TCacheSlot items. Do not use TCacheSlotArray directly, instead, use PCacheSlotArray (277) and allocate memory dynamically.

```
TOnFreeSlot = procedure(ACache: TCache; SlotIndex: Integer) of
  object
```

TOnFreeSlot is a callback prototype used when not enough slots are free, and a slot must be freed.

```
TOnIsDataEqual = function(ACache: TCache; AData1: Pointer;
  AData2: Pointer) : Boolean of object
```

TOnIsDataEqual is a callback prototype; It is used by the TCache.Add (279) call to determine whether the item to be added is a new item or not. The function returns True if the 2 data pointers AData1 and AData2 should be considered equal, or False when they are not.

For most purposes, comparing the pointers will be enough, but if the pointers are ansistrings, then the contents should be compared.

9.4 TCacheSlot

```
TCacheSlot = record
  Prev : PCacheSlot;
  Next : PCacheSlot;
  Data
  : Pointer;
  Index : Integer;
end
```

TCacheSlot is internally used by the TCache (278) class. It represents 1 element in the linked list.

9.5 ECacheError

9.5.1 Description

Exception class used in the cachecls unit.

9.6 TCache

9.6.1 Description

TCache implements a cache class: it is a list-like class, but which uses a counting mechanism, and keeps a Most-Recent-Used list; this list represents the 'cache'. The list is internally kept as a doubly-linked list.

The Data (281) property offers indexed access to the array of items. When accessing the array through this property, the MRUSlot (281) property is updated.

9.6.2 Method overview

Page	Method	Description
279	Add	Add a data element to the list.
280	AddNew	Add a new item to the list.
279	Create	Create a new cache class.
279	Destroy	Free the `TCache` class from memory.
280	FindSlot	Find data pointer in the list.
280	IndexOf	Return index of a data pointer in the list.
281	Remove	Remove a data item from the list.

9.6.3 Property overview

Page	Properties	Access	Description
281	Data	rw	Indexed access to data items.
282	LRUSlot	r	Last used item.
281	MRUSlot	rw	Most recent item slot.
283	OnFreeSlot	rw	Event called when a slot is freed.
282	OnIsDataEqual	rw	Event to compare 2 items.
282	SlotCount	rw	Number of slots in the list.
282	Slots	r	Indexed array to the slots.

9.6.4 TCache.Create

Synopsis: Create a new cache class.

Declaration:
```
constructor Create(ASlotCount: Integer)
```

Visibility: public

Description: `Create` instantiates a new instance of `TCache`. It allocates room for `ASlotCount` entries in the list. The number of slots can be increased later.

See also: TCache.SlotCount (282)

9.6.5 TCache.Destroy

Synopsis: Free the `TCache` class from memory.

Declaration:
```
destructor Destroy;  Override
```

Visibility: public

Description: `Destroy` cleans up the array for the elements, and calls the inherited `Destroy`. The elements in the array are not freed by this action.

See also: TCache.Create (279)

9.6.6 TCache.Add

Synopsis: Add a data element to the list.

Declaration:
```
function Add(AData: Pointer) : Integer
```

Visibility: public

Description: `Add` checks whether `AData` is already in the list. If so, the item is added to the top of the MRU list. If the item is not yet in the list, then the item is added to the list and placed at the top of the MRU list using the AddNew (280) call.

The function returns the index at which the item was added.

If the maximum number of slots is reached, and a new item is being added, the least used item is dropped from the list.

See also: TCache.AddNew (280), TCache.FindSlot (280), TCache.IndexOf (280), TCache.Data (281), TCache.MRUSlot (281)

9.6.7 TCache.AddNew

Synopsis: Add a new item to the list.

Declaration: `function AddNew(AData: Pointer) : Integer`

Visibility: public

Description: `AddNew` adds a new item to the list: in difference with the Add (279) call, no checking is performed to see whether the item is already in the list.

The function returns the index at which the item was added.

If the maximum number of slots is reached, and a new item is being added, the least used item is dropped from the list.

See also: TCache.Add (279), TCache.FindSlot (280), TCache.IndexOf (280), TCache.Data (281), TCache.MRUSlot (281)

9.6.8 TCache.FindSlot

Synopsis: Find data pointer in the list.

Declaration: `function FindSlot(AData: Pointer) : PCacheSlot`

Visibility: public

Description: `FindSlot` checks all items in the list, and returns the slot which contains a data pointer that matches the pointer `AData`.

If no item with data pointer that matches `AData` is found, `Nil` is returned.

For this function to work correctly, the OnIsDataEqual (282) event must be set.

Errors: If `OnIsDataEqual` is not set, an exception will be raised.

See also: TCache.IndexOf (280), TCache.Add (279), TCache.OnIsDataEqual (282)

9.6.9 TCache.IndexOf

Synopsis: Return index of a data pointer in the list.

Declaration: `function IndexOf(AData: Pointer) : Integer`

Visibility: public

Description: `IndexOF` searches in the list for a slot with data pointer that matches `AData` and returns the index of the slot.

If no item with data pointer that matches `AData` is found, `-1` is returned.

For this function to work correctly, the OnIsDataEqual (282) event must be set.

Errors: If `OnIsDataEqual` is not set, an exception will be raised.

See also: TCache.FindSlot (280), TCache.Add (279), TCache.OnIsDataEqual (282)

9.6.10 TCache.Remove

Synopsis: Remove a data item from the list.

Declaration: `procedure Remove(AData: Pointer)`

Visibility: public

Description: `Remove` searches the slot which matches `AData` and if it is found, sets the data pointer to `Nil`, thus effectively removing the pointer from the list.

Errors: None.

See also: TCache.FindSlot (280)

9.6.11 TCache.Data

Synopsis: Indexed access to data items.

Declaration: `Property Data[SlotIndex: Integer]: Pointer`

Visibility: public

Access: Read,Write

Description: `Data` offers index-based access to the data pointers in the cache. By accessing an item in the list in this manner, the item is moved to the front of the MRU list, i.e. MRUSlot (281) will point to the accessed item. The access is both read and write.

The index is zero-based and can maximally be SlotCount-1 (282). Providing an invalid index will result in an exception.

See also: TCache.MRUSlot (281)

9.6.12 TCache.MRUSlot

Synopsis: Most recent item slot.

Declaration: `Property MRUSlot : PCacheSlot`

Visibility: public

Access: Read,Write

Description: `MRUSlot` points to the most recent used slot. The most recent used slot is updated when the list is accessed through the Data (281) property, or when an item is added to the list with Add (279) or AddNew (280)

See also: TCache.Add (279), TCache.AddNew (280), TCache.Data (281), TCache.LRUSlot (282)

9.6.13 TCache.LRUSlot

Synopsis: Last used item.

Declaration: `Property LRUSlot : PCacheSlot`

Visibility: public

Access: Read

Description: `LRUSlot` points to the least recent used slot. It is the last item in the chain of slots.

See also: TCache.Add (279), TCache.AddNew (280), TCache.Data (281), TCache.MRUSlot (281)

9.6.14 TCache.SlotCount

Synopsis: Number of slots in the list.

Declaration: `Property SlotCount : Integer`

Visibility: public

Access: Read,Write

Description: `SlotCount` is the number of slots in the list. Its initial value is set when the `TCache` instance is created, but this can be changed at any time. If items are added to the list and the list is full, then the number of slots is not increased, but the least used item is dropped from the list. In that case OnFreeSlot (283) is called.

See also: TCache.Create (279), TCache.Data (281), TCache.Slots (282)

9.6.15 TCache.Slots

Synopsis: Indexed array to the slots.

Declaration: `Property Slots[SlotIndex: Integer]: PCacheSlot`

Visibility: public

Access: Read

Description: `Slots` provides index-based access to the `TCacheSlot` records in the list. Accessing the records directly does not change their position in the MRU list.

The index is zero-based and can maximally be SlotCount-1 (282). Providing an invalid index will result in an exception.

See also: TCache.Data (281), TCache.SlotCount (282)

9.6.16 TCache.OnIsDataEqual

Synopsis: Event to compare 2 items.

Declaration: `Property OnIsDataEqual : TOnIsDataEqual`

Visibility: public

Access: Read,Write

Description: `OnIsDataEqual` is used by FindSlot (280) and IndexOf (280) to compare items when looking for a particular item. These functions are called by the Add (279) method. Failing to set this event will result in an exception. The function should return `True` if the 2 data pointers should be considered equal.

See also: TCache.FindSlot (280), TCache.IndexOf (280), TCache.Add (279)

9.6.17 TCache.OnFreeSlot

Synopsis: Event called when a slot is freed.

Declaration: `Property OnFreeSlot : TOnFreeSlot`

Visibility: public

Access: Read,Write

Description: `OnFreeSlot` is called when an item needs to be freed, i.e. when a new item is added to a full list, and the least recent used item needs to be dropped from the list.

The cache class instance and the index of the item to be removed are passed to the callback.

See also: TCache.Add (279), TCache.AddNew (280), TCache.SlotCount (282)

Chapter 10

Reference for unit 'Contnrs'

10.1 Used units

Table 10.1: Used units by unit 'Contnrs'

Name	Page
Classes	**??**
System	**??**
sysutils	**??**

10.2 Overview

The contnrs unit implements various general-purpose classes:

Object lists lists that manage objects instead of pointers, and which automatically dispose of the objects.

Component lists lists that manage components instead of pointers, and which automatically dispose the components.

Class lists lists that manage class pointers instead of pointers.

Stacks Stack classes to push/pop pointers or objects

Queues Classes to manage a FIFO list of pointers or objects

Hash lists General-purpose Hash lists.

10.3 Constants, types and variables

10.3.1 Constants

```
MaxHashListSize = Maxint div 16
```

`MaxHashListSize` is the maximum number of elements a hash list can contain.

```
MaxHashStrSize = Maxint
```

MaxHashStrSize is the maximum amount of data for the key string values. The key strings are kept in a continuous memory area. This constant determines the maximum size of this memory area.

```
MaxHashTableSize = Maxint div 4
```

MaxHashTableSize is the maximum number of elements in the hash.

```
MaxItemsPerHash = 3
```

MaxItemsPerHash is the threshold above which the hash is expanded. If the number of elements in a hash bucket becomes larger than this value, the hash size is increased.

```
ViEmpty = 0
```

```
ViRealIndexOffset = 1
```

10.3.2 Types

```
PBucket = ^TBucket
```

Pointer to TBucket (289)" type.

```
PViHashListItem = ^TFPHashListItem
```

```
TBucketArray = Array of TBucket
```

Array of TBucket (289) records.

```
TBucketItemArray = Array of TBucketItem
```

Array of TBucketItem records.

```
TBucketListSizes = (bl2,bl4,bl8,bl16,bl32,bl64,bl128,bl256)
```

Table 10.2: Enumeration values for type TBucketListSizes

Value	Explanation
bl128	List with 128 buckets.
bl16	List with 16 buckets.
bl2	List with 2 buckets.
bl256	List with 256 buckets.
bl32	List with 32 buckets.
bl4	List with 4 buckets.
bl64	List with 64 buckets.
bl8	List with 8 buckets.

TBucketListSizes is used to set the bucket list size: It specified the number of buckets created by TBucketList (289).

```
TBucketProc = procedure(AInfo: Pointer; AItem: Pointer; AData: Pointer
  ;
                        out AContinue: Boolean)
```

TBucketProc is the prototype for the TCustomBucketList.Foreach (298) call. It is the plain procedural form. The Continue parameter can be set to False to indicate that the Foreach call should stop the iteration.

For a procedure of object (a method) callback, see the TBucketProcObject (286) prototype.

```
TBucketProcObject = procedure(AItem: Pointer; AData: Pointer;
  out AContinue: Boolean) of object
```

TBucketProcObject is the prototype for the TCustomBucketList.Foreach (298) call. It is the method (procedure of object) form. The Continue parameter can be set to False to indicate that the Foreach call should stop the iteration.

For a plain procedural callback, see the TBucketProc (286) prototype.

```
TDataIteratorCallBack = procedure(Item: Pointer; const Key: AnsiString
  ;
                                  var Continue: Boolean)
```

TDataIteratorCallBack is a callback prototype for the TFPDataHashTable.Iterate (304) static CallBack. It is called for each data pointer in the hash list, passing the key (key) and data pointer (item) for each item in the list. If Continue is set to false, the iteration stops.

```
TDataIteratorMethod = procedure(Item: Pointer; const Key: AnsiString
  ;
                                var Continue: Boolean) of
  object
```

TDataIteratorMethod is a callback prototype for the TFPDataHashTable.Iterate (304) method. It is called for each data pointer in the hash list, passing the key (key) and data pointer (item) for each item in the list. If Continue is set to false, the iteration stops.

```
THashFunction = function(const S: AnsiString; const TableSize: LongWord
  )
                          : LongWord
```

THashFunction is the prototype for a hash calculation function. It should calculate a hash of string S, where the hash table size is TableSize. The return value should be the hash value.

```
THTCustomNodeClass = Class of THTCustomNode
```

THTCustomNodeClass was used by TFPCustomHashTable (298) to decide which class should be created for elements in the list.

```
THTNode = THTDataNode
```

THTNode is provided for backwards compatibility.

```
TIteratorMethod = TDataIteratorMethod
```

TIteratorMethod is used in an internal TFPDataHashTable (304) method.

```
TObjectIteratorCallback = procedure(Item: TObject;
  const Key: AnsiString;
  var Continue: Boolean)
```

TObjectIteratorCallBack is the iterator callback prototype. It is used to iterate over all items in the hash table, and is called with each key value (Key) and associated object (Item). If Continue is set to false, the iteration stops.

```
TObjectIteratorMethod = procedure(Item: TObject; const Key: AnsiString
  ;
                                  var Continue: Boolean) of
  object
```

TObjectIteratorMethod is the iterator callback prototype. It is used to iterate over all items in the hash table, and is called with each key value (Key) and associated object (Item). If Continue is set to false, the iteration stops.

```
TObjectListCallback = procedure(data: TObject; arg: pointer) of
  object
```

TObjectListCallback is used as the prototype for the TFPObjectList.ForEachCall (329) link callwhen a method should be called. The Data argument will contain each of the objects in the list in turn, and the Data argument will contain the data passed to the ForEachCall call.

```
TObjectListStaticCallback = procedure(data: TObject; arg: pointer
  )
```

TObjectListCallback is used as the prototype for the TFPObjectList.ForEachCall (329) link call when a plain procedure should be called. The Data argument will contain each of the objects in the list in turn, and the Data argument will contain the data passed to the ForEachCall call.

```
TStringIteratorCallback = procedure(Item: AnsiString;
  const Key: AnsiString;
  var Continue: Boolean)
```

TStringIteratorCallBack is the callback prototype for the TFPStringHashTable (331) method. It is called for each element in the hash table, with the string. If Continue is set to False, the iteration stops.

```
TStringIteratorMethod = procedure(Item: AnsiString;
  const Key: AnsiString;
  var Continue: Boolean) of object
```

TStringIteratorMethod is the callback prototype for the TFPStringHashTable (331) method. It is called for each element in the hash table, with the string. If Continue is set to False, the iteration stops.

```
TViRehashMode = (vi_Auto,vi_Tight,vi_Pack)
```

Table 10.3: Enumeration values for type TViRehashMode

Value	Explanation
vi_Auto	
vi_Pack	
vi_Tight	

10.4 Procedures and functions

10.4.1 RSHash

Synopsis: Standard hash value calculating function.

Declaration:
```
function RSHash(const S: AnsiString; const TableSize: LongWord)
                    : LongWord
```

Visibility: default

Description: `RSHash` is the standard hash calculating function used in the TFPCustomHashTable (298) hash class. It's Robert Sedgwick's "Algorithms in C" hash function.

Errors: None.

See also: TFPCustomHashTable (298)

10.4.2 ViDataSize

Declaration:
```
function ViDataSize(n: SizeUint; bitsPerIndex: SizeUint) : SizeUint
```

Visibility: default

10.4.3 ViGet

Declaration:
```
function ViGet(data: PSizeUint; index: SizeUint; bitsPerIndex: SizeUint)
                    : SizeUint
```

Visibility: default

10.4.4 ViSet

Declaration:
```
procedure ViSet(data: PSizeUint; index: SizeUint;
                    bitsPerIndex: SizeUint; value: SizeUint)
```

Visibility: default

10.5 TBucket

```
TBucket = record
  Count : Integer;
  Items : TBucketItemArray;
end
```

TBucket describes 1 bucket in the TCustomBucketList (296) class. It is a container for TBucketItem (289) records. It should never be used directly.

10.6 TBucketItem

```
TBucketItem = record
  Item : Pointer;
  Data : Pointer;
end
```

TBucketItem is a record used for internal use in TCustomBucketList (296). It should not be necessary to use it directly.

10.7 TFPHashListItem

```
TFPHashListItem = record
  HashValue : uint32;
  Next : int32;
  Str
  : RawByteString;
  Data : Pointer;
end
```

10.8 EDuplicate

10.8.1 Description

Exception raised when a key is stored twice in a hash table.

10.9 EKeyNotFound

10.9.1 Description

Exception raised when a key is not found.

See also: TFPCustomHashTable.Delete (301)

10.10 TBucketList

10.10.1 Description

TBucketList is a descendent of TCustomBucketList which allows to specify a bucket count which is a multiple of 2, up to 256 buckets. The size is passed to the constructor and cannot be changed in the lifetime of the bucket list instance.

The buckets for an item is determined by looking at the last bits of the item pointer: For 2 buckets, the last bit is examined, for 4 buckets, the last 2 bits are taken and so on. The algorithm takes into account the average granularity (4) of heap pointers.

See also: TCustomBucketList (296)

10.10.2 Method overview

Page	Method	Description
290	Create	Create a new `TBucketList` instance.

10.10.3 TBucketList.Create

Synopsis: Create a new `TBucketList` instance.

Declaration: `constructor Create(ABuckets: TBucketListSizes)`

Visibility: public

Description: `Create` instantiates a new bucketlist instance with a number of buckets determined by `ABuckets`. After creation, the number of buckets can no longer be changed.

Errors: If not enough memory is available to create the instance, an exception may be raised.

See also: TBucketListSizes (285)

10.11 TClassList

10.11.1 Description

`TClassList` is a Tlist (**??**) descendent which stores class references instead of pointers. It introduces no new behaviour other than ensuring all stored pointers are class pointers.

The `OwnsObjects` property as found in `TComponentList` and `TObjectList` is not implemented as there are no actual instances.

See also: #rtl.classes.tlist (**??**), TComponentList (293), TObjectList (336)

10.11.2 Method overview

Page	Method	Description
291	Add	Add a new class pointer to the list.
291	Extract	Extract a class pointer from the list.
292	First	Returns the first non-nil class pointer.
291	IndexOf	Search for a class pointer in the list.
292	Insert	Insert a new class pointer in the list.
292	Last	Return last non-`Nil` class pointer.
291	Remove	Remove a class pointer from the list.

10.11.3 Property overview

Page	Properties	Access	Description
292	Items	rw	Index based access to class pointers.

10.11.4 TClassList.Add

Synopsis: Add a new class pointer to the list.

Declaration: `function Add(AClass: TClass) : Integer`

Visibility: public

Description: `Add` adds `AClass` to the list, and returns the position at which it was added. It simply overrides the TList (**??**) bevahiour, and introduces no new functionality.

Errors: If not enough memory is available to expand the list, an exception may be raised.

See also: TClassList.Extract (291), #rtl.classes.tlist.add (**??**)

10.11.5 TClassList.Extract

Synopsis: Extract a class pointer from the list.

Declaration: `function Extract(Item: TClass) : TClass`

Visibility: public

Description: `Extract` extracts a class pointer `Item` from the list, if it is present in the list. It returns the extracted class pointer, or `Nil` if the class pointer was not present in the list. It simply overrides the implementation in `TList` so it accepts a class pointer instead of a simple pointer. No new behaviour is introduced.

Errors: None.

See also: TClassList.Remove (291), #rtl.classes.Tlist.Extract (**??**)

10.11.6 TClassList.Remove

Synopsis: Remove a class pointer from the list.

Declaration: `function Remove(AClass: TClass) : Integer`

Visibility: public

Description: `Remove` removes a class pointer `Item` from the list, if it is present in the list. It returns the index of the removed class pointer, or `-1` if the class pointer was not present in the list. It simply overrides the implementation in `TList` so it accepts a class pointer instead of a simple pointer. No new behaviour is introduced.

Errors: None.

See also: TClassList.Extract (291), #rtl.classes.Tlist.Remove (**??**)

10.11.7 TClassList.IndexOf

Synopsis: Search for a class pointer in the list.

Declaration: `function IndexOf(AClass: TClass) : Integer`

Visibility: public

Description: `IndexOf` searches for `AClass` in the list, and returns it's position if it was found, or -1 if it was not found in the list.

Errors: None.

See also: #rtl.classes.tlist.indexof (**??**)

10.11.8 TClassList.First

Synopsis: Returns the first non-nil class pointer.

Declaration: `function First : TClass`

Visibility: public

Description: `First` returns a reference to the first non-`Nil` class pointer in the list. If no non-`Nil` element is found, `Nil` is returned. `Nil` is returned.

See also: TClassList.Last (292)

10.11.9 TClassList.Last

Synopsis: Return last non-`Nil` class pointer.

Declaration: `function Last : TClass`

Visibility: public

Description: `Last` returns a reference to the last non-`Nil` class pointer in the list. If no non-`Nil` element is found, `Nil` is returned.

See also: TClassList.First (292)

10.11.10 TClassList.Insert

Synopsis: Insert a new class pointer in the list.

Declaration: `procedure Insert(Index: Integer; AClass: TClass)`

Visibility: public

Description: `Insert` inserts a class pointer in the list at position `Index`. It simply overrides the parent implementation so it only accepts class pointers. It introduces no new behaviour.

Errors: None.

See also: #rtl.classes.TList.Insert (**??**), TClassList.Add (291), TClassList.Remove (291)

10.11.11 TClassList.Items

Synopsis: Index based access to class pointers.

Declaration: `Property Items[Index: Integer]: TClass; default`

Visibility: public

Access: Read,Write

Description: `Items` provides index-based access to the class pointers in the list. `TClassList` overrides the default `Items` implementation of `TList` so it returns class pointers instead of pointers.

See also: #rtl.classes.TList.Items (**??**), #rtl.classes.TList.Count (**??**)

10.12 TComponentList

10.12.1 Description

`TComponentList` is a TObjectList (336) descendent which has as the default array property TComponents (**??**) instead of objects. It overrides some methods so only components can be added.

In difference with TObjectList (336), `TComponentList` removes any `TComponent` from the list if the `TComponent` instance was freed externally. It uses the `FreeNotification` mechanism for this.

See also: #rtl.classes.TList (**??**), TFPObjectList (323), TObjectList (336), TClassList (290)

10.12.2 Method overview

Page	Method	Description
293	Add	Add a component to the list.
293	Destroy	Destroys the instance.
294	Extract	Remove a component from the list without destroying it.
295	First	First non-nil instance in the list.
294	IndexOf	Search for an instance in the list.
295	Insert	Insert a new component in the list.
295	Last	Last non-nil instance in the list.
294	Remove	Remove a component from the list, possibly destroying it.

10.12.3 Property overview

Page	Properties	Access	Description
295	Items	rw	Index-based access to the elements in the list.

10.12.4 TComponentList.Destroy

Synopsis: Destroys the instance.

Declaration:
```
destructor Destroy;  Override
```

Visibility: public

Description: `Destroy` unhooks the free notification handler and then calls the inherited destroy to clean up the `TComponentList` instance.

Errors: None.

See also: TObjectList (336), #rtl.classes.TComponent (**??**)

10.12.5 TComponentList.Add

Synopsis: Add a component to the list.

Declaration:
```
function Add(AComponent: TComponent) : Integer
```

Visibility: public

Description: `Add` overrides the `Add` operation of it's ancestors, so it only accepts `TComponent` instances. It introduces no new behaviour.

The function returns the index at which the component was added.

Errors: If not enough memory is available to expand the list, an exception may be raised.

See also: TObjectList.Add (337)

10.12.6 TComponentList.Extract

Synopsis: Remove a component from the list without destroying it.

Declaration: `function Extract(Item: TComponent) : TComponent`

Visibility: public

Description: `Extract` removes a component (`Item`) from the list, without destroying it. It overrides the implementation of TObjectList (336) so only `TComponent` descendents can be extracted. It introduces no new behaviour.

`Extract` returns the instance that was extracted, or `Nil` if no instance was found.

See also: TComponentList.Remove (294), TObjectList.Extract (337)

10.12.7 TComponentList.Remove

Synopsis: Remove a component from the list, possibly destroying it.

Declaration: `function Remove(AComponent: TComponent) : Integer`

Visibility: public

Description: `Remove` removes `item` from the list, and if the list owns it's items, it also destroys it. It returns the index of the item that was removed, or -1 if no item was removed.

`Remove` simply overrides the implementation in TObjectList (336) so it only accepts `TComponent` descendents. It introduces no new behaviour.

Errors: None.

See also: TComponentList.Extract (294), TObjectList.Remove (337)

10.12.8 TComponentList.IndexOf

Synopsis: Search for an instance in the list.

Declaration: `function IndexOf(AComponent: TComponent) : Integer`

Visibility: public

Description: `IndexOf` searches for an instance in the list and returns it's position in the list. The position is zero-based. If no instance is found, -1 is returned.

`IndexOf` just overrides the implementation of the parent class so it accepts only `TComponent` instances. It introduces no new behaviour.

Errors: None.

See also: TObjectList.IndexOf (338)

10.12.9 TComponentList.First

Synopsis: First non-nil instance in the list.

Declaration: `function First : TComponent`

Visibility: public

Description: `First` overrides the implementation of it's ancestors to return the first non-nil instance of `TComponent` in the list. If no non-nil instance is found, `Nil` is returned.

Errors: None.

See also: TComponentList.Last (295), TObjectList.First (338)

10.12.10 TComponentList.Last

Synopsis: Last non-nil instance in the list.

Declaration: `function Last : TComponent`

Visibility: public

Description: `Last` overrides the implementation of it's ancestors to return the last non-nil instance of `TComponent` in the list. If no non-nil instance is found, `Nil` is returned.

Errors: None.

See also: TComponentList.First (295), TObjectList.Last (339)

10.12.11 TComponentList.Insert

Synopsis: Insert a new component in the list.

Declaration: `procedure Insert(Index: Integer; AComponent: TComponent)`

Visibility: public

Description: `Insert` inserts a `TComponent` instance (`AComponent`) in the list at position `Index`. It simply overrides the parent implementation so it only accepts `TComponent` instances. It introduces no new behaviour.

Errors: None.

See also: TObjectList.Insert (338), TComponentList.Add (293), TComponentList.Remove (294)

10.12.12 TComponentList.Items

Synopsis: Index-based access to the elements in the list.

Declaration: `Property Items[Index: Integer]: TComponent; default`

Visibility: public

Access: Read,Write

Description: `Items` provides access to the components in the list using an index. It simply overrides the default property of the parent classes so it returns/accepts `TComponent` instances only. Note that the index is zero based.

See also: TObjectList.Items (339)

10.13 TCustomBucketList

10.13.1 Description

`TCustomBucketList` is an associative list using buckets for storage. It scales better than a regular TList (**??**) list class, especially when an item must be searched in the list.

Since the list associates a data pointer with each item pointer, it follows that each item pointer must be unique, and can be added to the list only once.

The `TCustomBucketList` class does not determine the number of buckets or the bucket hash mechanism, this must be done by descendent classes such as TBucketList (289). `TCustomBucketList` only takes care of storage and retrieval of items in the various buckets.

Because `TCustomBucketList` is an abstract class - it does not determine the number of buckets - one should never instantiate an instance of `TCustomBucketList`, but always use a descendent class such as TCustomBucketList (296).

See also: TBucketList (289)

10.13.2 Method overview

Page	Method	Description
297	Add	Add an item to the list.
297	Assign	Assign one bucket list to another.
296	Clear	Clear the list.
296	Destroy	Frees the bucketlist from memory.
297	Exists	Check if an item exists in the list.
297	Find	Find an item in the list.
298	ForEach	Loop over all items.
298	Remove	Remove an item from the list.

10.13.3 Property overview

Page	Properties	Access	Description
298	Data	rw	Associative array for data pointers.

10.13.4 TCustomBucketList.Destroy

Synopsis: Frees the bucketlist from memory.

Declaration: `destructor Destroy;  Override`

Visibility: public

Description: `Destroy` frees all storage for the buckets from memory. The items themselves are not freed from memory.

10.13.5 TCustomBucketList.Clear

Synopsis: Clear the list.

Declaration: `procedure Clear`

Visibility: public

Description: `Clear` clears the list. The items and their data themselves are not disposed of, this must be done separately. Clear only removes all references to the items from the list.

Errors: None.

See also: TCustomBucketList.Add (297)

10.13.6 TCustomBucketList.Add

Synopsis: Add an item to the list.

Declaration: `function Add(AItem: Pointer; AData: Pointer) : Pointer`

Visibility: public

Description: `Add` adds `AItem` with it's associated `AData` to the list and returns `AData`.

Errors: If `AItem` is already in the list, an `EListError` exception will be raised.

See also: TCustomBucketList.Exists (297), TCustomBucketList.Clear (296)

10.13.7 TCustomBucketList.Assign

Synopsis: Assign one bucket list to another.

Declaration: `procedure Assign(AList: TCustomBucketList)`

Visibility: public

Description: `Assign` is implemented by `TCustomBucketList` to copy the contents of another bucket list to the bucket list. It clears the contents prior to the copy operation.

See also: TCustomBucketList.Add (297), TCustomBucketList.Clear (296)

10.13.8 TCustomBucketList.Exists

Synopsis: Check if an item exists in the list.

Declaration: `function Exists(AItem: Pointer) : Boolean`

Visibility: public

Description: `Exists` searches the list and returns `True` if the `AItem` is already present in the list. If the item is not yet in the list, `False` is returned.

If the data pointer associated with `AItem` is also needed, then it is better to use Find (297).

See also: TCustomBucketList.Find (297)

10.13.9 TCustomBucketList.Find

Synopsis: Find an item in the list.

Declaration: `function Find(AItem: Pointer; out AData: Pointer) : Boolean`

Visibility: public

Description: `Find` searches for `AItem` in the list and returns the data pointer associated with it in `AData` if the item was found. In that case the return value is `True`. If `AItem` is not found in the list, `False` is returned.

See also: TCustomBucketList.Exists (297)

10.13.10 TCustomBucketList.ForEach

Synopsis: Loop over all items.

Declaration:
```
function ForEach(AProc: TBucketProc; AInfo: Pointer) : Boolean
function ForEach(AProc: TBucketProcObject) : Boolean
```

Visibility: public

Description: `Foreach` loops over all items in the list and calls `AProc`, passing it in turn each item in the list.

`AProc` exists in 2 variants: one which is a simple procedure, and one which is a method. In the case of the simple procedure, the `AInfo` argument is passed as well in each call to `AProc`.

The loop stops when all items have been processed, or when the `AContinue` argument of `AProc` contains `False` on return.

The result of the function is `True` if all items were processed, or `False` if the loop was interrupted with a `AContinue` return of `False`.

Errors: None.

See also: TCustomBucketList.Data (298)

10.13.11 TCustomBucketList.Remove

Synopsis: Remove an item from the list.

Declaration:
```
function Remove(AItem: Pointer) : Pointer
```

Visibility: public

Description: `Remove` removes `AItem` from the list, and returns the associated data pointer of the removed item. If the item was not in the list, then `Nil` is returned.

See also: Find (297)

10.13.12 TCustomBucketList.Data

Synopsis: Associative array for data pointers.

Declaration:
```
Property Data[AItem: Pointer]: Pointer; default
```

Visibility: public

Access: Read,Write

Description: `Data` provides direct access to the `Data` pointers associated with the `AItem` pointers. If `AItem` is not in the list of pointers, an `EListError` exception will be raised.

See also: TCustomBucketList.Find (297), TCustomBucketList.Exists (297)

10.14 TFPCustomHashTable

10.14.1 Description

`TFPCustomHashTable` is a general-purpose hashing class. It can store string keys and pointers associated with these strings. The hash mechanism is configurable and can be optionally be specified

when a new instance of the class is created; A default hash mechanism is implemented in RSHash (288).

The TFPHashList (305) can also be used when fast lookup of data based on some key is required. It is slightly faster than the `TFPCustomHashTable` implementation, but the keys are limited to a length of 256 characters, and it is not suitable for re-use: it is a one-time fill, many times search object. `TFPCustomHashTable` is slower, but handles re-use better.

See also: THTCustomNode (331), TFPObjectList (323), RSHash (288)

10.14.2 Method overview

Page	Method	Description
300	ChangeTableSize	Change the table size of the hash table.
300	Clear	Clear the hash table.
299	Create	Instantiate a new `TFPCustomHashTable` instance using the default hash mechanism.
300	CreateWith	Instantiate a new `TFPCustomHashTable` instance with given algorithm and size.
301	Delete	Delete a key from the hash list.
300	Destroy	Free the hash table.
301	Find	Search for an item with a certain key value.
301	IsEmpty	Check if the hash table is empty.

10.14.3 Property overview

Page	Properties	Access	Description
303	AVGChainLen	r	Average chain length.
302	Count	r	Number of items in the hash table.
304	Density	r	Number of filled slots.
301	HashFunction	rw	Hash function currently in use.
302	HashTable	r	Hash table instance.
302	HashTableSize	rw	Size of the hash table.
303	LoadFactor	r	Fraction of count versus size.
303	MaxChainLength	r	Maximum chain length.
303	NumberOfCollisions	r	Number of extra items.
302	VoidSlots	r	Number of empty slots in the hash table.

10.14.4 TFPCustomHashTable.Create

Synopsis: Instantiate a new `TFPCustomHashTable` instance using the default hash mechanism.

Declaration: `constructor Create`

Visibility: public

Description: `Create` creates a new instance of `TFPCustomHashTable` with hash size 196613 and hash algorithm RSHash (288)

Errors: If no memory is available, an exception may be raised.

See also: CreateWith (300)

10.14.5 TFPCustomHashTable.CreateWith

Synopsis: Instantiate a new `TFPCustomHashTable` instance with given algorithm and size.

Declaration:
```
constructor CreateWith(AHashTableSize: LongWord;
                       aHashFunc: THashFunction)
```

Visibility: public

Description: `CreateWith` creates a new instance of `TFPCustomHashTable` with hash size `AHashTableSize` and hash calculating algorithm `aHashFunc`.

Errors: If no memory is available, an exception may be raised.

See also: Create (299)

10.14.6 TFPCustomHashTable.Destroy

Synopsis: Free the hash table.

Declaration:
```
destructor Destroy;  Override
```

Visibility: public

Description: `Destroy` removes the hash table from memory. If any data was associated with the keys in the hash table, then this data is not freed. This must be done by the programmer.

See also: Destroy (300), Create (299), CreateWith (300)

10.14.7 TFPCustomHashTable.ChangeTableSize

Synopsis: Change the table size of the hash table.

Declaration:
```
procedure ChangeTableSize(const ANewSize: LongWord);  Virtual
```

Visibility: public

Description: `ChangeTableSize` changes the size of the hash table: it recomputes the hash value for all of the keys in the table, so this is an expensive operation.

Errors: If no memory is available, an exception may be raised.

See also: HashTableSize (302)

10.14.8 TFPCustomHashTable.Clear

Synopsis: Clear the hash table.

Declaration:
```
procedure Clear;  Virtual
```

Visibility: public

Description: `Clear` removes all keys and their associated data from the hash table. The data itself is not freed from memory, this should be done by the programmer.

Errors: None.

See also: Destroy (300)

10.14.9 TFPCustomHashTable.Delete

Synopsis: Delete a key from the hash list.

Declaration: `procedure Delete(const aKey: AnsiString);  Virtual`

Visibility: public

Description: `Delete` deletes all keys with value `AKey` from the hash table. It does not free the data associated with key. If `AKey` is not in the list, nothing is removed.

Errors: None.

See also: TFPCustomHashTable.Find (301)

10.14.10 TFPCustomHashTable.Find

Synopsis: Search for an item with a certain key value.

Declaration: `function Find(const aKey: AnsiString) : THTCustomNode`

Visibility: public

Description: `Find` searches for the THTCustomNode (331) instance with key value equal to `Akey` and if it finds it, it returns the instance. If no matching value is found, `Nil` is returned.

Note that the instance returned by this function cannot be freed; If it should be removed from the hash table, the Delete (301) method should be used instead.

Errors: None.

See also: Delete (301)

10.14.11 TFPCustomHashTable.IsEmpty

Synopsis: Check if the hash table is empty.

Declaration: `function IsEmpty : Boolean`

Visibility: public

Description: `IsEmpty` returns `True` if the hash table contains no elements, or `False` if there are still elements in the hash table.

See also: TFPCustomHashTable.Count (302), TFPCustomHashTable.HashTableSize (302), TFPCustomHashTable.AVGChainLen (303), TFPCustomHashTable.MaxChainLength (303)

10.14.12 TFPCustomHashTable.HashFunction

Synopsis: Hash function currently in use.

Declaration: `Property HashFunction : THashFunction`

Visibility: public

Access: Read,Write

Description: `HashFunction` is the hash function currently in use to calculate hash values from keys. The property can be set, this simply calls `SetHashFunction`. Note that setting the hash function does **NOT** cause the hash value of all keys to be recomputed, so changing the value while there are still keys in the table is not a good idea.

See also: HashTableSize (302)

10.14.13 TFPCustomHashTable.Count

Synopsis: Number of items in the hash table.

Declaration: `Property Count : LongWord`

Visibility: public

Access: Read

Description: `Count` is the number of items in the hash table.

See also: TFPCustomHashTable.IsEmpty (301), TFPCustomHashTable.HashTableSize (302), TFPCustomHashTable.AVGChainLen (303), TFPCustomHashTable.MaxChainLength (303)

10.14.14 TFPCustomHashTable.HashTableSize

Synopsis: Size of the hash table.

Declaration: `Property HashTableSize : LongWord`

Visibility: public

Access: Read,Write

Description: `HashTableSize` is the size of the hash table. It can be set, in which case it will be rounded to the nearest prime number suitable for RSHash.

See also: TFPCustomHashTable.IsEmpty (301), TFPCustomHashTable.Count (302), TFPCustomHashTable.AVGChainLen (303), TFPCustomHashTable.MaxChainLength (303), TFPCustomHashTable.VoidSlots (302), TFPCustomHashTable.Density (304)

10.14.15 TFPCustomHashTable.HashTable

Synopsis: Hash table instance.

Declaration: `Property HashTable : TFPObjectList`

Visibility: public

Access: Read

Description: `TFPCustomHashTable` is the internal list object (TFPObjectList (323) used for the hash table. Each element in this table is again a TFPObjectList (323) instance or `Nil`.

10.14.16 TFPCustomHashTable.VoidSlots

Synopsis: Number of empty slots in the hash table.

Declaration: `Property VoidSlots : LongWord`

Visibility: public

Access: Read

Description: `VoidSlots` is the number of empty slots in the hash table. Calculating this is an expensive operation.

See also: TFPCustomHashTable.IsEmpty (301), TFPCustomHashTable.Count (302), TFPCustomHashTable.AVGChainLen (303), TFPCustomHashTable.MaxChainLength (303), TFPCustomHashTable.LoadFactor (303), TFPCustomHashTable.Density (304), TFPCustomHashTable.NumberOfCollisions (303)

10.14.17 TFPCustomHashTable.LoadFactor

Synopsis: Fraction of count versus size.

Declaration: `Property LoadFactor : Double`

Visibility: public

Access: Read

Description: `LoadFactor` is the ratio of elements in the table versus table size. Ideally, this should be as small as possible.

See also: TFPCustomHashTable.IsEmpty (301), TFPCustomHashTable.Count (302), TFPCustomHashTable.AVGChainLen (303), TFPCustomHashTable.MaxChainLength (303), TFPCustomHashTable.VoidSlots (302), TFPCustomHashTable.Density (304), TFPCustomHashTable.NumberOfCollisions (303)

10.14.18 TFPCustomHashTable.AVGChainLen

Synopsis: Average chain length.

Declaration: `Property AVGChainLen : Double`

Visibility: public

Access: Read

Description: `AVGChainLen` is the average chain length, i.e. the ratio of elements in the table versus the number of filled slots. Calculating this is an expensive operation.

See also: TFPCustomHashTable.IsEmpty (301), TFPCustomHashTable.Count (302), TFPCustomHashTable.LoadFactor (303), TFPCustomHashTable.MaxChainLength (303), TFPCustomHashTable.VoidSlots (302), TFPCustomHashTable.Density (304), TFPCustomHashTable.NumberOfCollisions (303)

10.14.19 TFPCustomHashTable.MaxChainLength

Synopsis: Maximum chain length.

Declaration: `Property MaxChainLength : LongWord`

Visibility: public

Access: Read

Description: `MaxChainLength` is the length of the longest chain in the hash table. Calculating this is an expensive operation.

See also: TFPCustomHashTable.IsEmpty (301), TFPCustomHashTable.Count (302), TFPCustomHashTable.LoadFactor (303), TFPCustomHashTable.AVGChainLen (303), TFPCustomHashTable.VoidSlots (302), TFPCustomHashTable.Density (304), TFPCustomHashTable.NumberOfCollisions (303)

10.14.20 TFPCustomHashTable.NumberOfCollisions

Synopsis: Number of extra items.

Declaration: `Property NumberOfCollisions : LongWord`

Visibility: public

Access: Read

Description: `NumberOfCollisions` is the number of items which are not the first item in a chain. If this number is too big, the hash size may be too small.

See also: TFPCustomHashTable.IsEmpty (301), TFPCustomHashTable.Count (302), TFPCustomHashTable.LoadFactor (303), TFPCustomHashTable.AVGChainLen (303), TFPCustomHashTable.VoidSlots (302), TFPCustomHashTable.Density (304)

10.14.21 TFPCustomHashTable.Density

Synopsis: Number of filled slots.

Declaration: `Property Density : LongWord`

Visibility: public

Access: Read

Description: `Density` is the number of filled slots in the hash table.

See also: TFPCustomHashTable.IsEmpty (301), TFPCustomHashTable.Count (302), TFPCustomHashTable.LoadFactor (303), TFPCustomHashTable.AVGChainLen (303), TFPCustomHashTable.VoidSlots (302), TFPCustomHashTable.Density (304)

10.15 TFPDataHashTable

10.15.1 Description

`TFPDataHashTable` is a TFPCustomHashTable (298) descendent which stores simple data pointers together with the keys. In case the data associated with the keys are objects, it's better to use TFPObjectHashTable (320), or for string data, TFPStringHashTable (330) is more suitable. The data pointers are exposed with their keys through the Items (305) property.

See also: TFPObjectHashTable (320), TFPStringHashTable (330), Items (305)

10.15.2 Method overview

Page	Method	Description
305	Add	Add a data pointer to the list.
304	Iterate	Iterate over the pointers in the hash table.

10.15.3 Property overview

Page	Properties	Access	Description
305	Items	rw	Key-based access to the items in the table.

10.15.4 TFPDataHashTable.Iterate

Synopsis: Iterate over the pointers in the hash table.

Declaration:
```
function Iterate(aMethod: TDataIteratorMethod) : Pointer;  Virtual
function Iterate(aMethod: TDataIteratorCallBack) : Pointer;  Virtual
```

Visibility: public

Description: `Iterate` iterates over all elements in the array, calling `aMethod` for each pointer, or until the method returns `False` in its `continue` parameter. It returns `Nil` if all elements were processed, or the pointer that was being processed when `aMethod` returned `False` in the `Continue` parameter.

The `aMethod` callback can be a method of an object, or a normal, static procedure.

10.15.5 TFPDataHashTable.Add

Synopsis: Add a data pointer to the list.

Declaration: `procedure Add(const aKey: AnsiString; AItem: pointer);  Virtual`

Visibility: public

Description: `Add` adds a data pointer (`AItem`) to the list with key `AKey`.

Errors: If `AKey` already exists in the table, an exception is raised.

See also: TFPDataHashTable.Items (305)

10.15.6 TFPDataHashTable.Items

Synopsis: Key-based access to the items in the table.

Declaration: `Property Items[index: AnsiString]: Pointer; default`

Visibility: public

Access: Read,Write

Description: `Items` provides access to the items in the hash table using their key: the array index `Index` is the key. A key which is not present will result in an `Nil` pointer.

See also: TFPStringHashTable.Add (331)

10.16 TFPHashList

10.16.1 Description

`TFPHashList` implements a fast hash class. The class is built for speed, therefore the key values can be shortstrings only, and the data can only be non-nil pointers.

if a base class for an own hash class is wanted, the TFPCustomHashTable (298) class can be used. If a hash class for objects is needed instead of pointers, the TFPHashObjectList (314) class can be used.

See also: TFPCustomHashTable (298), TFPHashObjectList (314), TFPDataHashTable (304), TFPStringHashTable (330)

10.16.2 Method overview

Page	Method	Description
307	Add	Add a new key/data pair to the list.
307	Clear	Clear the list.
306	Create	Create a new instance of the hashlist.
308	Delete	Delete an item from the list.
306	Destroy	Removes an instance of the hashlist from the heap.
308	Error	Raise an error.
308	Extract	Extract a pointer from the list.
309	Find	Find data associated with key.
309	FindIndexOf	Return index of named item.
309	FindWithHash	Find first element with given name and hash value.
310	ForEachCall	Call a procedure for each element in the list.
308	GetNextCollision	Get next collision number.
307	HashOfIndex	Return the hash value of an item by index.
308	IndexOf	Return the index of the data pointer.
307	NameOfIndex	Returns the key name of an item by index.
310	Pack	Remove nil pointers from the list.
310	Remove	Remove first instance of a pointer.
310	Rename	Rename a key.

10.16.3 Property overview

Page	Properties	Access	Description
311	Capacity	rw	Capacity of the list.
310	Count	r	Current number of elements in the list.
311	Items	rw	Indexed array with pointers.
311	List	r	Low-level hash list.

10.16.4 TFPHashList.Create

Synopsis: Create a new instance of the hashlist.

Declaration: `constructor Create`

Visibility: public

Description: `Create` creates a new instance of `TFPHashList` on the heap and sets the hash capacity to 1.

See also: TFPHashList.Destroy (306)

10.16.5 TFPHashList.Destroy

Synopsis: Removes an instance of the hashlist from the heap.

Declaration: `destructor Destroy;  Override`

Visibility: public

Description: `Destroy` cleans up the memory structures maintained by the hashlist and removes the `TFPHashList` instance from the heap.

`Destroy` should not be called directly, it's better to use `Free` or `FreeAndNil` instead.

See also: TFPHashList.Create (306), TFPHashList.Clear (307)

10.16.6 TFPHashList.Add

Synopsis: Add a new key/data pair to the list.

Declaration: `function Add(const AName: RawByteString; Item: Pointer) : SizeInt`

Visibility: public

Description: `Add` adds a new data pointer (`Item`) with key `AName` to the list. It returns the position of the item in the list.

Errors: If not enough memory is available to hold the key and data, an exception may be raised.

See also: TFPHashList.Extract (308), TFPHashList.Remove (310), TFPHashList.Delete (308)

10.16.7 TFPHashList.Clear

Synopsis: Clear the list.

Declaration: `procedure Clear`

Visibility: public

Description: `Clear` removes all items from the list. It does not free the data items themselves. It frees all memory needed to contain the items.

Errors: None.

See also: TFPHashList.Extract (308), TFPHashList.Remove (310), TFPHashList.Delete (308), TFPHashList.Add (307)

10.16.8 TFPHashList.NameOfIndex

Synopsis: Returns the key name of an item by index.

Declaration: `function NameOfIndex(Index: SizeInt) : RawByteString`

Visibility: public

Description: `NameOfIndex` returns the key name of the item at position `Index`.

Errors: If `Index` is out of the valid range, an exception is raised.

See also: TFPHashList.HashOfIndex (307), TFPHashList.Find (309), TFPHashList.FindIndexOf (309), TFPHashList.FindWithHash (309)

10.16.9 TFPHashList.HashOfIndex

Synopsis: Return the hash value of an item by index.

Declaration: `function HashOfIndex(Index: SizeInt) : LongWord`

Visibility: public

Description: `HashOfIndex` returns the hash value of the item at position `Index`.

Errors: If `Index` is out of the valid range, an exception is raised.

See also: TFPHashList.NameOfIndex (307), TFPHashList.Find (309), TFPHashList.FindIndexOf (309), TFPHashList.FindWithHash (309)

10.16.10 TFPHashList.GetNextCollision

Synopsis: Get next collision number.

Declaration: `function GetNextCollision(Index: SizeInt) : SizeInt`

Visibility: public

Description: `GetNextCollision` returns the next collision in hash item `Index`. This is the count of items with the same hash.means that the next it

10.16.11 TFPHashList.Delete

Synopsis: Delete an item from the list.

Declaration: `procedure Delete(Index: SizeInt)`

Visibility: public

Description: `Delete` deletes the item at position `Index`. The data to which it points is not freed from memory.

Errors: TFPHashList.Extract (308)TFPHashList.Remove (310)TFPHashList.Add (307)

10.16.12 TFPHashList.Error

Synopsis: Raise an error.

Declaration: `class procedure Error(const Msg: AnsiString; Data: PtrInt)`

Visibility: public

Description: `Error` raises an `EListError` exception, with message `Msg`. The `Data` pointer is used to format the message.

10.16.13 TFPHashList.Extract

Synopsis: Extract a pointer from the list.

Declaration: `function Extract(item: Pointer) : Pointer`

Visibility: public

Description: `Extract` removes the data item from the list, if it is in the list. It returns the pointer if it was removed from the list, `Nil` otherwise.

`Extract` does a linear search, and is not very efficient.

See also: TFPHashList.Delete (308), TFPHashList.Remove (310), TFPHashList.Clear (307)

10.16.14 TFPHashList.IndexOf

Synopsis: Return the index of the data pointer.

Declaration: `function IndexOf(Item: Pointer) : SizeInt`

Visibility: public

Description: `IndexOf` returns the index of the first occurrence of pointer `Item`. If the item is not in the list, -1 is returned.

The performed search is linear, and not very efficient.

See also: TFPHashList.HashOfIndex (307), TFPHashList.NameOfIndex (307), TFPHashList.Find (309), TFPHashList.FindIndexOf (309), TFPHashList.FindWithHash (309)

10.16.15 TFPHashList.Find

Synopsis: Find data associated with key.

Declaration: `function Find(const AName: RawByteString) : Pointer`

Visibility: public

Description: `Find` searches (using the hash) for the data item associated with item `AName` and returns the data pointer associated with it. If the item is not found, `Nil` is returned. It uses the hash value of the key to perform the search.

See also: TFPHashList.HashOfIndex (307), TFPHashList.NameOfIndex (307), TFPHashList.IndexOf (308), TFPHashList.FindIndexOf (309), TFPHashList.FindWithHash (309)

10.16.16 TFPHashList.FindIndexOf

Synopsis: Return index of named item.

Declaration: `function FindIndexOf(const AName: RawByteString) : SizeInt`

Visibility: public

Description: `FindIndexOf` returns the index of the key `AName`, or -1 if the key does not exist in the list. It uses the hash value to search for the key. Note that `Nil` data pointers will result in -1 as well.

See also: TFPHashList.HashOfIndex (307), TFPHashList.NameOfIndex (307), TFPHashList.IndexOf (308), TFPHashList.Find (309), TFPHashList.FindWithHash (309)

10.16.17 TFPHashList.FindWithHash

Synopsis: Find first element with given name and hash value.

Declaration:
```
function FindWithHash(const AName: RawByteString; AHash: LongWord)
                      : Pointer
```

Visibility: public

Description: `FindWithHash` searches for the item with key `AName`. It uses the provided hash value `AHash` to perform the search. If the item exists, the data pointer is returned, if not, the result is `Nil`.

See also: TFPHashList.HashOfIndex (307), TFPHashList.NameOfIndex (307), TFPHashList.IndexOf (308), TFPHashList.Find (309), TFPHashList.FindIndexOf (309)

10.16.18 TFPHashList.Rename

Synopsis: Rename a key.

Declaration:
```
function Rename(const AOldName: RawByteString;
                const ANewName: RawByteString) : SizeInt
```

Visibility: public

Description: `Rename` renames key `AOldname` to `ANewName`. The hash value is recomputed and the item is moved in the list to it's new position.

Errors: If an item with `ANewName` already exists, an exception will be raised.

10.16.19 TFPHashList.Remove

Synopsis: Remove first instance of a pointer.

Declaration:
```
function Remove(Item: Pointer) : SizeInt
```

Visibility: public

Description: `Remove` removes the first occurrence of the data pointer `Item` in the list, if it is present. The return value is the removed data pointer, or `Nil` if no data pointer was removed.

See also: TFPHashList.Delete (308), TFPHashList.Clear (307), TFPHashList.Extract (308)

10.16.20 TFPHashList.Pack

Synopsis: Remove nil pointers from the list.

Declaration:
```
procedure Pack
```

Visibility: public

Description: `Pack` removes all `Nil` items from the list, and frees all unused memory.

See also: TFPHashList.Clear (307)

10.16.21 TFPHashList.ForEachCall

Synopsis: Call a procedure for each element in the list.

Declaration:
```
procedure ForEachCall(proc2call: TListCallback; arg: pointer)
procedure ForEachCall(proc2call: TListStaticCallback; arg: pointer)
```

Visibility: public

Description: `ForEachCall` loops over the items in the list and calls `proc2call`, passing it the item and `arg`.

10.16.22 TFPHashList.Count

Synopsis: Current number of elements in the list.

Declaration:
```
Property Count : int32
```

Visibility: public

Access: Read

Description: `Count` is the current number of elements in the list.

See also: Capacity (311), Items (311)

10.16.23 TFPHashList.Capacity

Synopsis: Capacity of the list.

Declaration: `Property Capacity : uint32`

Visibility: public

Access: Read,Write

Description: `Capacity` returns the current capacity of the list. The capacity is expanded as more elements are added to the list. If a good estimate of the number of elements that will be added to the list, the property can be set to a sufficiently large value to avoid reallocation of memory each time the list needs to grow.

See also: Count (310), Items (311)

10.16.24 TFPHashList.Items

Synopsis: Indexed array with pointers.

Declaration: `Property Items[Index: SizeInt]: Pointer; default`

Visibility: public

Access: Read,Write

Description: `Items` provides indexed access to the pointers, the index runs from 0 to Count-1 (310).

Errors: Specifying an invalid index will result in an exception.

See also: Capacity (311), Count (310)

10.16.25 TFPHashList.List

Synopsis: Low-level hash list.

Declaration: `Property List : PViHashListItem`

Visibility: public

Access: Read

Description: `List` exposes the low-level item list (284). It should not be used directly.

See also: Strs (305), THashItemList (284)

10.17 TFPHashObject

10.17.1 Description

`TFPHashObject` is a `TObject` descendent which is aware of the TFPHashObjectList (314) class. It has a name property and an owning list: if the name is changed, it will reposition itself in the list which owns it. It offers methods to change the owning list: the object will correctly remove itself from the list which currently owns it, and insert itself in the new list.

See also: TFPHashObject.Name (313), TFPHashObject.ChangeOwner (313), TFPHashObject.ChangeOwnerAndName (313)

10.17.2 Method overview

Page	Method	Description
313	ChangeOwner	Change the list owning the object.
313	ChangeOwnerAndName	Simultaneously change the list owning the object and the name of the object.
312	Create	Create a named instance, and insert in a hash list.
312	CreateNotOwned	Create an instance not owned by any list.
313	Rename	Rename the object.

10.17.3 Property overview

Page	Properties	Access	Description
314	Hash	r	Hash value.
313	Name	r	Current name of the object.
314	OwnerList	r	

10.17.4 TFPHashObject.CreateNotOwned

Synopsis: Create an instance not owned by any list.

Declaration:
```
constructor CreateNotOwned
```

Visibility: public

Description: `CreateNotOwned` creates an instance of `TFPHashObject` which is not owned by any TFPHashObjectList (314) hash list. It also has no name when created in this way.

See also: TFPHashObject.Name (313), TFPHashObject.ChangeOwner (313), TFPHashObject.ChangeOwnerAndName (313)

10.17.5 TFPHashObject.Create

Synopsis: Create a named instance, and insert in a hash list.

Declaration:
```
constructor Create(HashObjectList: TFPHashObjectList;
                   const s: RawByteString)
```

Visibility: public

Description: `Create` creates an instance of `TFPHashObject`, gives it the name `S` and inserts it in the hash list HashObjectList (314).

See also: CreateNotOwned (312), TFPHashObject.ChangeOwner (313), TFPHashObject.Name (313)

10.17.6 TFPHashObject.ChangeOwner

Synopsis: Change the list owning the object.

Declaration:
```
procedure ChangeOwner(HashObjectList: TFPHashObjectList)
```

Visibility: public

Description: `ChangeOwner` can be used to move the object between hash lists: The object will be removed correctly from the hash list that currently owns it, and will be inserted in the list `HashObjectList`.

Errors: If an object with the same name already is present in the new hash list, an exception will be raised.

See also: ChangeOwnerAndName (313), Name (313)

10.17.7 TFPHashObject.ChangeOwnerAndName

Synopsis: Simultaneously change the list owning the object and the name of the object.

Declaration:
```
procedure ChangeOwnerAndName(HashObjectList: TFPHashObjectList;
                             const s: RawByteString)
```

Visibility: public

Description: `ChangeOwnerAndName` can be used to move the object between hash lists: The object will be removed correctly from the hash list that currently owns it (using the current name), and will be inserted in the list `HashObjectList` with the new name `S`.

Errors: If the new name already is present in the new hash list, an exception will be raised.

See also: ChangeOwner (313), Name (313)

10.17.8 TFPHashObject.Rename

Synopsis: Rename the object.

Declaration:
```
procedure Rename(const ANewName: RawByteString)
```

Visibility: public

Description: `Rename` changes the name of the object, and notifies the hash list of this change.

Errors: If the new name already is present in the hash list, an exception will be raised.

See also: ChangeOwner (313), ChangeOwnerAndName (313), Name (313)

10.17.9 TFPHashObject.Name

Synopsis: Current name of the object.

Declaration:
```
Property Name : RawByteString
```

Visibility: public

Access: Read

Description: `Name` is the name of the object, it is stored in the hash list using this name as the key.

See also: Rename (313), ChangeOwnerAndName (313)

10.17.10 TFPHashObject.Hash

Synopsis: Hash value.

Declaration: `Property Hash : LongWord`

Visibility: public

Access: Read

Description: `Hash` is the hash value of the object in the hash list that owns it.

See also: Name (313)

10.17.11 TFPHashObject.OwnerList

Declaration: `Property OwnerList : TFPHashObjectList`

Visibility: public

Access: Read

10.18 TFPHashObjectList

10.18.1 Method overview

Page	Method	Description
315	Add	Add a new key/data pair to the list.
315	Clear	Clear the list.
315	Create	Create a new instance of the hashlist.
316	Delete	Delete an object from the list.
315	Destroy	Removes an instance of the hashlist from the heap.
317	Extract	Extract a object instance from the list.
317	Find	Find data associated with key.
318	FindIndexOf	Return index of named object.
318	FindInstanceOf	Search an instance of a certain class.
318	FindWithHash	Find first element with given name and hash value.
319	ForEachCall	Call a procedure for each object in the list.
316	GetNextCollision	Get next collision number.
316	HashOfIndex	Return the hash value of an object by index.
317	IndexOf	Return the index of the object instance.
316	NameOfIndex	Returns the key name of an object by index.
319	Pack	Remove nil object instances from the list.
317	Remove	Remove first occurrence of a object instance.
318	Rename	Rename a key.

10.18.2 Property overview

Page	Properties	Access	Description
319	Capacity	rw	Capacity of the list.
319	Count	r	Current number of elements in the list.
320	Items	rw	Indexed array with object instances.
320	List	r	Low-level hash list.
320	OwnsObjects	rw	Does the list own the objects it contains.

10.18.3 TFPHashObjectList.Create

Synopsis: Create a new instance of the hashlist.

Declaration: `constructor Create(FreeObjects: Boolean)`

Visibility: public

Description: `Create` creates a new instance of `TFPHashObjectList` on the heap and sets the hash capacity to 1.

If `FreeObjects` is `True` (the default), then the list owns the objects: when an object is removed from the list, it is destroyed (freed from memory). Clearing the list will free all objects in the list.

See also: TFPHashObjectList.Destroy (315), TFPHashObjectList.OwnsObjects (320)

10.18.4 TFPHashObjectList.Destroy

Synopsis: Removes an instance of the hashlist from the heap.

Declaration: `destructor Destroy;  Override`

Visibility: public

Description: `Destroy` cleans up the memory structures maintained by the hashlist and removes the `TFPHashObjectList` instance from the heap. If the list owns its objects, they are freed from memory as well.

`Destroy` should not be called directly, it's better to use `Free` or `FreeAndNil` instead.

See also: TFPHashObjectList.Create (315), TFPHashObjectList.Clear (315)

10.18.5 TFPHashObjectList.Clear

Synopsis: Clear the list.

Declaration: `procedure Clear`

Visibility: public

Description: `Clear` removes all objects from the list. It does not free the objects themselves, unless OwnsObjects (320) is `True`. It always frees all memory needed to contain the objects.

Errors: None.

See also: TFPHashObjectList.Extract (317), TFPHashObjectList.Remove (317), TFPHashObjectList.Delete (316), TFPHashObjectList.Add (315)

10.18.6 TFPHashObjectList.Add

Synopsis: Add a new key/data pair to the list.

Declaration: `function Add(const AName: RawByteString; AObject: TObject) : Integer`

Visibility: public

Description: `Add` adds a new object instance (`AObject`) with key `AName` to the list. It returns the position of the object in the list.

Errors: If not enough memory is available to hold the key and data, an exception may be raised. If an object with this name already exists in the list, an exception is raised.

See also: TFPHashObjectList.Extract (317), TFPHashObjectList.Remove (317), TFPHashObjectList.Delete (316)

10.18.7 TFPHashObjectList.NameOfIndex

Synopsis: Returns the key name of an object by index.

Declaration: `function NameOfIndex(Index: Integer) : RawByteString`

Visibility: public

Description: `NameOfIndex` returns the key name of the object at position `Index`.

Errors: If `Index` is out of the valid range, an exception is raised.

See also: TFPHashObjectList.HashOfIndex (316), TFPHashObjectList.Find (317), TFPHashObjectList.FindIndexOf (318), TFPHashObjectList.FindWithHash (318)

10.18.8 TFPHashObjectList.HashOfIndex

Synopsis: Return the hash value of an object by index.

Declaration: `function HashOfIndex(Index: Integer) : LongWord`

Visibility: public

Description: `HashOfIndex` returns the hash value of the object at position `Index`.

Errors: If `Index` is out of the valid range, an exception is raised.

See also: TFPHashObjectList.NameOfIndex (316), TFPHashObjectList.Find (317), TFPHashObjectList.FindIndexOf (318), TFPHashObjectList.FindWithHash (318)

10.18.9 TFPHashObjectList.GetNextCollision

Synopsis: Get next collision number.

Declaration: `function GetNextCollision(Index: Integer) : Integer`

Visibility: public

Description: Get next collision number.

10.18.10 TFPHashObjectList.Delete

Synopsis: Delete an object from the list.

Declaration: `procedure Delete(Index: Integer)`

Visibility: public

Description: `Delete` deletes the object at position `Index`. If OwnsObjects (320) is `True`, then the object itself is also freed from memory.

See also: TFPHashObjectList.Extract (317), TFPHashObjectList.Remove (317), TFPHashObjectList.Add (315), OwnsObjects (320)

10.18.11 TFPHashObjectList.Extract

Synopsis: Extract a object instance from the list.

Declaration: `function Extract(Item: TObject) : TObject`

Visibility: public

Description: `Extract` removes the data object from the list, if it is in the list. It returns the object instance if it was removed from the list, `Nil` otherwise. The object is *not* freed from memory, regardless of the value of OwnsObjects (320).

`Extract` does a linear search, and is not very efficient.

See also: TFPHashObjectList.Delete (316), TFPHashObjectList.Remove (317), TFPHashObjectList.Clear (315)

10.18.12 TFPHashObjectList.Remove

Synopsis: Remove first occurrence of a object instance.

Declaration: `function Remove(AObject: TObject) : Integer`

Visibility: public

Description: `Remove` removes the first occurrence of the object instance `Item` in the list, if it is present. The return value is the location of the removed object instance, or `-1` if no object instance was removed.

If OwnsObjects (320) is `True`, then the object itself is also freed from memory.

See also: TFPHashObjectList.Delete (316), TFPHashObjectList.Clear (315), TFPHashObjectList.Extract (317)

10.18.13 TFPHashObjectList.IndexOf

Synopsis: Return the index of the object instance.

Declaration: `function IndexOf(AObject: TObject) : Integer`

Visibility: public

Description: `IndexOf` returns the index of the first occurrence of object instance `AObject`. If the object is not in the list, -1 is returned.

The performed search is linear, and not very efficient.

See also: TFPHashObjectList.HashOfIndex (316), TFPHashObjectList.NameOfIndex (316), TFPHashObjectList.Find (317), TFPHashObjectList.FindIndexOf (318), TFPHashObjectList.FindWithHash (318)

10.18.14 TFPHashObjectList.Find

Synopsis: Find data associated with key.

Declaration: `function Find(const s: RawByteString) : TObject`

Visibility: public

Description: `Find` searches (using the hash) for the data object associated with key `AName` and returns the data object instance associated with it. If the object is not found, `Nil` is returned. It uses the hash value of the key to perform the search.

See also: TFPHashObjectList.HashOfIndex (316), TFPHashObjectList.NameOfIndex (316), TFPHashObjectList.IndexOf (317), TFPHashObjectList.FindIndexOf (318), TFPHashObjectList.FindWithHash (318)

10.18.15 TFPHashObjectList.FindIndexOf

Synopsis: Return index of named object.

Declaration:
```
function FindIndexOf(const s: RawByteString) : Integer
```

Visibility: public

Description: `FindIndexOf` returns the index of the key `AName`, or -1 if the key does not exist in the list. It uses the hash value to search for the key.

See also: TFPHashObjectList.HashOfIndex (316), TFPHashObjectList.NameOfIndex (316), TFPHashObjectList.IndexOf (317), TFPHashObjectList.Find (317), TFPHashObjectList.FindWithHash (318)

10.18.16 TFPHashObjectList.FindWithHash

Synopsis: Find first element with given name and hash value.

Declaration:
```
function FindWithHash(const AName: RawByteString; AHash: LongWord)
                      : Pointer
```

Visibility: public

Description: `FindWithHash` searches for the object with key `AName`. It uses the provided hash value `AHash` to perform the search. If the object exists, the data object instance is returned, if not, the result is `Nil`.

See also: TFPHashObjectList.HashOfIndex (316), TFPHashObjectList.NameOfIndex (316), TFPHashObjectList.IndexOf (317), TFPHashObjectList.Find (317), TFPHashObjectList.FindIndexOf (318)

10.18.17 TFPHashObjectList.Rename

Synopsis: Rename a key.

Declaration:
```
function Rename(const AOldName: RawByteString;
                const ANewName: RawByteString) : Integer
```

Visibility: public

Description: `Rename` renames key `AOldname` to `ANewName`. The hash value is recomputed and the object is moved in the list to it's new position.

Errors: If an object with `ANewName` already exists, an exception will be raised.

10.18.18 TFPHashObjectList.FindInstanceOf

Synopsis: Search an instance of a certain class.

Declaration:
```
function FindInstanceOf(AClass: TClass; AExact: Boolean;
                        AStartAt: Integer) : Integer
```

Visibility: public

Description: `FindInstanceOf` searches the list for an instance of class `AClass`. It starts searching at position `AStartAt`. If `AExact` is `True`, only instances of class `AClass` are considered. If `AExact` is `False`, then descendent classes of `AClass` are also taken into account when searching. If no instance is found, `Nil` is returned.

10.18.19 TFPHashObjectList.Pack

Synopsis: Remove nil object instances from the list.

Declaration:
```
procedure Pack
```

Visibility: public

Description: `Pack` removes all `Nil` objects from the list, and frees all unused memory.

See also: TFPHashObjectList.Clear (315)

10.18.20 TFPHashObjectList.ForEachCall

Synopsis: Call a procedure for each object in the list.

Declaration:
```
procedure ForEachCall(proc2call: TObjectListCallback; arg: pointer)
procedure ForEachCall(proc2call: TObjectListStaticCallback;
                      arg: pointer)
```

Visibility: public

Description: `ForEachCall` loops over the objects in the list and calls `proc2call`, passing it the object and `arg`.

10.18.21 TFPHashObjectList.Capacity

Synopsis: Capacity of the list.

Declaration:
```
Property Capacity : Integer
```

Visibility: public

Access: Read,Write

Description: `Capacity` returns the current capacity of the list. The capacity is expanded as more elements are added to the list. If a good estimate of the number of elements that will be added to the list, the property can be set to a sufficiently large value to avoid reallocation of memory each time the list needs to grow.

See also: Count (319), Items (320)

10.18.22 TFPHashObjectList.Count

Synopsis: Current number of elements in the list.

Declaration:
```
Property Count : Integer
```

Visibility: public

Access: Read

Description: `Count` is the current number of elements in the list.

See also: Capacity (319), Items (320)

10.18.23 TFPHashObjectList.OwnsObjects

Synopsis: Does the list own the objects it contains.

Declaration: `Property OwnsObjects : Boolean`

Visibility: public

Access: Read,Write

Description: `OwnsObjects` determines what to do when an object is removed from the list: if it is `True` (the default), then the list owns the objects: when an object is removed from the list, it is destroyed (freed from memory). Clearing the list will free all objects in the list.

The value of `OwnsObjects` is set when the hash list is created, and may not be changed during the lifetime of the hash list. (The property is made read-only in versions later than 3.0 of Free Pascal).

See also: TFPHashObjectList.Create (315)

10.18.24 TFPHashObjectList.Items

Synopsis: Indexed array with object instances.

Declaration: `Property Items[Index: Integer]: TObject; default`

Visibility: public

Access: Read,Write

Description: `Items` provides indexed access to the object instances, the index runs from 0 to Count-1 (319).

Errors: Specifying an invalid index will result in an exception.

See also: Capacity (319), Count (319)

10.18.25 TFPHashObjectList.List

Synopsis: Low-level hash list.

Declaration: `Property List : TFPHashList`

Visibility: public

Access: Read

Description: `List` exposes the low-level hash list (305). It should not be used directly.

See also: TFPHashList (305)

10.19 TFPObjectHashTable

10.19.1 Description

`TFPStringHashTable` is a TFPCustomHashTable (298) descendent which stores object instances together with the keys. In case the data associated with the keys are strings themselves, it's better to use TFPStringHashTable (330), or for arbitrary pointer data, TFPDataHashTable (304) is more suitable. The objects are exposed with their keys through the Items (322) property.

See also: TFPStringHashTable (330), TFPDataHashTable (304), TFPObjectHashTable.Items (322)

10.19.2 Method overview

Page	Method	Description
322	Add	Add a new object to the hash table.
321	Create	Create a new instance of `TFPObjectHashTable`.
321	CreateWith	Create a new hash table with given size and hash function.
322	Iterate	Iterate over the objects in the hash table.

10.19.3 Property overview

Page	Properties	Access	Description
322	Items	rw	Key-based access to the objects.
322	OwnsObjects	r	Does the hash table own the objects ?

10.19.4 TFPObjectHashTable.Create

Synopsis: Create a new instance of `TFPObjectHashTable`.

Declaration:
```
constructor Create(AOwnsObjects: Boolean)
```

Visibility: public

Description: `Create` creates a new instance of `TFPObjectHashTable` on the heap. It sets the OwnsObjects (322) property to `AOwnsObjects`, and then calls the inherited `Create`. If `AOwnsObjects` is set to `True`, then the hash table owns the objects: whenever an object is removed from the list, it is automatically freed.

Errors: If not enough memory is available on the heap, an exception may be raised.

See also: TFPObjectHashTable.OwnsObjects (322), TFPObjectHashTable.CreateWith (321), TFPObjectHashTable.Items (322)

10.19.5 TFPObjectHashTable.CreateWith

Synopsis: Create a new hash table with given size and hash function.

Declaration:
```
constructor CreateWith(AHashTableSize: LongWord;
                       aHashFunc: THashFunction; AOwnsObjects: Boolean)
```

Visibility: public

Description: `CreateWith` sets the OwnsObjects (322) property to `AOwnsObjects`, and then calls the inherited `CreateWith`. If `AOwnsObjects` is set to `True`, then the hash table owns the objects: whenever an object is removed from the list, it is automatically freed.

This constructor should be used when a table size and hash algorithm should be specified that differ from the default table size and hash algorithm.

Errors: If not enough memory is available on the heap, an exception may be raised.

See also: TFPObjectHashTable.OwnsObjects (322), TFPObjectHashTable.Create (321), TFPObjectHashTable.Items (322)

10.19.6 TFPObjectHashTable.Iterate

Synopsis: Iterate over the objects in the hash table.

Declaration:
```
function Iterate(aMethod: TObjectIteratorMethod) : TObject;  Virtual
function Iterate(aMethod: TObjectIteratorCallback) : TObject;  Virtual
```

Visibility: public

Description: `Iterate` iterates over all elements in the array, calling `aMethod` for each object, or until the method returns `False` in its `continue` parameter. It returns `Nil` if all elements were processed, or the object that was being processed when `aMethod` returned `False` in the `Continue` parameter.

10.19.7 TFPObjectHashTable.Add

Synopsis: Add a new object to the hash table.

Declaration:
```
procedure Add(const aKey: AnsiString; AItem: TObject);  Virtual
```

Visibility: public

Description: `Add` adds the object `AItem` to the hash table, and associates it with key `aKey`.

Errors: If the key `aKey` is already in the hash table, an exception will be raised.

See also: TFPObjectHashTable.Items (322)

10.19.8 TFPObjectHashTable.Items

Synopsis: Key-based access to the objects.

Declaration:
```
Property Items[index: Ansistring]: TObject; default
```

Visibility: public

Access: Read,Write

Description: `Items` provides access to the objects in the hash table using their key: the array index `Index` is the key. A key which is not present will result in an `Nil` instance.

See also: TFPObjectHashTable.Add (322)

10.19.9 TFPObjectHashTable.OwnsObjects

Synopsis: Does the hash table own the objects ?

Declaration:
```
Property OwnsObjects : Boolean
```

Visibility: public

Access: Read

Description: `OwnsObjects` determines what happens with objects which are removed from the hash table: if `True`, then removing an object from the hash list will free the object. If `False`, the object is not freed. Note that way in which the object is removed is not relevant: be it `Delete`, `Remove` or `Clear`.

See also: TFPObjectHashTable.Create (321), TFPObjectHashTable.Items (322)

10.20 TFPObjectList

10.20.1 Description

`TFPObjectList` is a TFPList (**??**) based list which has as the default array property TObjects (**??**) instead of pointers. By default it also manages the objects: when an object is deleted or removed from the list, it is automatically freed. This behaviour can be disabled when the list is created.

In difference with TObjectList (336), `TFPObjectList` offers no notification mechanism of list operations, allowing it to be faster than `TObjectList`. For the same reason, it is also not a descendent of `TFPList` (although it uses one internally).

See also: #rtl.classes.TFPList (**??**), TObjectList (336)

10.20.2 Method overview

Page	Method	Description
324	Add	Add an object to the list.
328	Assign	Copy the contents of a list.
324	Clear	Clear all elements in the list.
323	Create	Create a new object list.
324	Delete	Delete an element from the list.
324	Destroy	Clears the list and destroys the list instance.
325	Exchange	Exchange the location of two objects.
325	Expand	Expand the capacity of the list.
325	Extract	Extract an object from the list.
326	FindInstanceOf	Search for an instance of a certain class.
327	First	Return the first non-nil object in the list.
329	ForEachCall	For each object in the list, call a method or procedure, passing it the object.
326	IndexOf	Search for an object in the list.
327	Insert	Insert a new object in the list.
327	Last	Return the last non-nil object in the list.
327	Move	Move an object to another location in the list.
328	Pack	Remove all `Nil` references from the list.
326	Remove	Remove an item from the list.
328	Sort	Sort the list of objects.

10.20.3 Property overview

Page	Properties	Access	Description
329	Capacity	rw	Capacity of the list.
329	Count	rw	Number of elements in the list.
330	Items	rw	Indexed access to the elements of the list.
330	List	r	Internal list used to keep the objects.
329	OwnsObjects	rw	Should the list free elements when they are removed.

10.20.4 TFPObjectList.Create

Synopsis: Create a new object list.

Declaration:
```
constructor Create
constructor Create(FreeObjects: Boolean)
```

Visibility: public

Description: `Create` instantiates a new object list. The `FreeObjects` parameter determines whether objects that are removed from the list should also be freed from memory. By default this is `True`. This behaviour can be changed after the list was instantiated.

Errors: None.

See also: TFPObjectList.Destroy (324), TFPObjectList.OwnsObjects (329), TObjectList (336)

10.20.5 TFPObjectList.Destroy

Synopsis: Clears the list and destroys the list instance.

Declaration: `destructor Destroy;  Override`

Visibility: public

Description: `Destroy` clears the list, freeing all objects in the list if OwnsObjects (329) is `True`.

See also: TFPObjectList.OwnsObjects (329), TObjectList.Create (336)

10.20.6 TFPObjectList.Clear

Synopsis: Clear all elements in the list.

Declaration: `procedure Clear`

Visibility: public

Description: Removes all objects from the list, freeing all objects in the list if OwnsObjects (329) is `True`.

10.20.7 TFPObjectList.Add

Synopsis: Add an object to the list.

Declaration: `function Add(AObject: TObject) : Integer`

Visibility: public

Description: `Add` adds `AObject` to the list and returns the index of the object in the list.

Note that when OwnsObjects (329) is `True`, an object should not be added twice to the list: this will result in memory corruption when the object is freed (as it will be freed twice). The `Add` method does not check this, however.

Errors: None.

See also: TFPObjectList.OwnsObjects (329), TFPObjectList.Delete (324)

10.20.8 TFPObjectList.Delete

Synopsis: Delete an element from the list.

Declaration: `procedure Delete(Index: Integer)`

Visibility: public

Description: `Delete` removes the object at index `Index` from the list. When OwnsObjects (329) is `True`, the object is also freed.

Errors: An access violation may occur when OwnsObjects (329) is `True` and either the object was freed externally, or when the same object is in the same list twice.

See also: TFPObjectList.Remove (326), TFPObjectList.Extract (325), TFPObjectList.OwnsObjects (329), TFPObjectList.Add (324), TFPObjectList.Clear (324)

10.20.9 TFPObjectList.Exchange

Synopsis: Exchange the location of two objects.

Declaration: `procedure Exchange(Index1: Integer; Index2: Integer)`

Visibility: public

Description: `Exchange` exchanges the objects at indexes `Index1` and `Index2` in a direct operation (i.e. no delete/add is performed).

Errors: If either `Index1` or `Index2` is invalid, an exception will be raised.

See also: TFPObjectList.Add (324), TFPObjectList.Delete (324)

10.20.10 TFPObjectList.Expand

Synopsis: Expand the capacity of the list.

Declaration: `function Expand : TFPObjectList`

Visibility: public

Description: `Expand` increases the capacity of the list. It calls #rtl.classes.tfplist.expand (**??**) and then returns a reference to itself.

Errors: If there is not enough memory to expand the list, an exception will be raised.

See also: TFPObjectList.Pack (328), TFPObjectList.Clear (324), #rtl.classes.tfplist.expand (**??**)

10.20.11 TFPObjectList.Extract

Synopsis: Extract an object from the list.

Declaration: `function Extract(Item: TObject) : TObject`

Visibility: public

Description: `Extract` removes `Item` from the list, if it is present in the list. It returns `Item` if it was found, `Nil` if item was not present in the list.

Note that the object is not freed, and that only the first found object is removed from the list.

Errors: None.

See also: TFPObjectList.Pack (328), TFPObjectList.Clear (324), TFPObjectList.Remove (326), TFPObjectList.Delete (324)

10.20.12 TFPObjectList.Remove

Synopsis: Remove an item from the list.

Declaration:
```
function Remove(AObject: TObject) : Integer
```

Visibility: public

Description: `Remove` removes `Item` from the list, if it is present in the list. It frees `Item` if OwnsObjects (329) is `True`, and returns the index of the object that was found in the list, or -1 if the object was not found.

Note that only the first found object is removed from the list.

Errors: None.

See also: TFPObjectList.Pack (328), TFPObjectList.Clear (324), TFPObjectList.Delete (324), TFPObjectList.Extract (325)

10.20.13 TFPObjectList.IndexOf

Synopsis: Search for an object in the list.

Declaration:
```
function IndexOf(AObject: TObject) : Integer
```

Visibility: public

Description: `IndexOf` searches for the presence of `AObject` in the list, and returns the location (index) in the list. The index is 0-based, and -1 is returned if `AObject` was not found in the list.

Errors: None.

See also: TFPObjectList.Items (330), TFPObjectList.Remove (326), TFPObjectList.Extract (325)

10.20.14 TFPObjectList.FindInstanceOf

Synopsis: Search for an instance of a certain class.

Declaration:
```
function FindInstanceOf(AClass: TClass; AExact: Boolean;
                        AStartAt: Integer) : Integer
```

Visibility: public

Description: `FindInstanceOf` will look through the instances in the list and will return the index of the first instance which is a descendent of class `AClass` if `AExact` is `False`. If `AExact` is true, then the instance should be of class `AClass`.

If no instance of the requested class is found, `-1` is returned.

Errors: None.

See also: TFPObjectList.IndexOf (326)

10.20.15 TFPObjectList.Insert

Synopsis: Insert a new object in the list.

Declaration: `procedure Insert(Index: Integer; AObject: TObject)`

Visibility: public

Description: `Insert` inserts `AObject` at position `Index` in the list. All elements in the list after this position are shifted. The index is zero based, i.e. an insert at position 0 will insert an object at the first position of the list.

Errors: None.

See also: TFPObjectList.Add (324), TFPObjectList.Delete (324)

10.20.16 TFPObjectList.First

Synopsis: Return the first non-nil object in the list.

Declaration: `function First : TObject`

Visibility: public

Description: `First` returns a reference to the first non-`Nil` element in the list. If no non-`Nil` element is found, `Nil` is returned.

Errors: None.

See also: TFPObjectList.Last (327), TFPObjectList.Pack (328)

10.20.17 TFPObjectList.Last

Synopsis: Return the last non-nil object in the list.

Declaration: `function Last : TObject`

Visibility: public

Description: `Last` returns a reference to the last non-`Nil` element in the list. If no non-`Nil` element is found, `Nil` is returned.

Errors: None.

See also: TFPObjectList.First (327), TFPObjectList.Pack (328)

10.20.18 TFPObjectList.Move

Synopsis: Move an object to another location in the list.

Declaration: `procedure Move(CurIndex: Integer; NewIndex: Integer)`

Visibility: public

Description: `Move` moves the object at current location `CurIndex` to location `NewIndex`. Note that the `NewIndex` is determined *after* the object was removed from location `CurIndex`, and can hence be shifted with 1 position if `CurIndex` is less than `NewIndex`.

Contrary to exchange (325), the move operation is done by extracting the object from it's current location and inserting it at the new location.

Errors: If either `CurIndex` or `NewIndex` is out of range, an exception may occur.

See also: TFPObjectList.Exchange (325), TFPObjectList.Delete (324), TFPObjectList.Insert (327)

10.20.19 TFPObjectList.Assign

Synopsis: Copy the contents of a list.

Declaration: `procedure Assign(Obj: TFPObjectList)`

Visibility: public

Description: `Assign` copies the contents of `Obj` if `Obj` is of type `TFPObjectList`

Errors: None.

10.20.20 TFPObjectList.Pack

Synopsis: Remove all `Nil` references from the list.

Declaration: `procedure Pack`

Visibility: public

Description: `Pack` removes all `Nil` elements from the list.

Errors: None.

See also: TFPObjectList.First (327), TFPObjectList.Last (327)

10.20.21 TFPObjectList.Sort

Synopsis: Sort the list of objects.

Declaration: `procedure Sort(Compare: TListSortCompare)`

Visibility: public

Description: `Sort` will perform a quick-sort on the list, using `Compare` as the compare algorithm. This function should accept 2 pointers and should return the following result:

less than 0 If the first pointer comes before the second.

equal to 0 If the pointers have the same value.

larger than 0 If the first pointer comes after the second.

The function should be able to deal with `Nil` values.

Errors: None.

See also: #rtl.classes.TList.Sort (**??**)

10.20.22 TFPObjectList.ForEachCall

Synopsis: For each object in the list, call a method or procedure, passing it the object.

Declaration:
```
procedure ForEachCall(proc2call: TObjectListCallback; arg: pointer)
procedure ForEachCall(proc2call: TObjectListStaticCallback;
                      arg: pointer)
```

Visibility: public

Description: `ForEachCall` loops through all objects in the list, and calls `proc2call`, passing it the object in the list. Additionally, `arg` is also passed to the procedure. `Proc2call` can be a plain procedure or can be a method of a class.

Errors: None.

See also: TObjectListStaticCallback (287), TObjectListCallback (287)

10.20.23 TFPObjectList.Capacity

Synopsis: 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 before it needs to expand itself, i.e., reserve more memory for pointers. It is always equal or larger than Count (329).

See also: TFPObjectList.Count (329)

10.20.24 TFPObjectList.Count

Synopsis: 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. Note that this includes `Nil` elements.

See also: TFPObjectList.Capacity (329)

10.20.25 TFPObjectList.OwnsObjects

Synopsis: Should the list free elements when they are removed.

Declaration:
```
Property OwnsObjects : Boolean
```

Visibility: public

Access: Read,Write

Description: `OwnsObjects` determines whether the objects in the list should be freed when they are removed (not extracted) from the list, or when the list is cleared. If the property is `True` then they are freed. If the property is `False` the elements are not freed.

The value is usually set in the constructor, and is seldom changed during the lifetime of the list. It defaults to `True`.

See also: TFPObjectList.Create (323), TFPObjectList.Delete (324), TFPObjectList.Remove (326), TFPObjectList.Clear (324)

10.20.26 TFPObjectList.Items

Synopsis: Indexed access to the elements of the list.

Declaration: `Property Items[Index: Integer]: TObject; default`

Visibility: public

Access: Read,Write

Description: `Items` is the default property of the list. It provides indexed access to the elements in the list. The index `Index` is zero based, i.e., runs from 0 (zero) to `Count-1`.

See also: TFPObjectList.Count (329)

10.20.27 TFPObjectList.List

Synopsis: Internal list used to keep the objects.

Declaration: `Property List : TFPList`

Visibility: public

Access: Read

Description: `List` is a reference to the TFPList (**??**) instance used to manage the elements in the list.

See also: #rtl.classes.tfplist (**??**)

10.21 TFPStringHashTable

10.21.1 Description

`TFPStringHashTable` is a TFPCustomHashTable (298) descendent which stores simple strings together with the keys. In case the data associated with the keys are objects, it's better to use TFPObjectHashTable (320), or for arbitrary pointer data, TFPDataHashTable (304) is more suitable. The strings are exposed with their keys through the Items (331) property.

See also: TFPObjectHashTable (320), TFPDataHashTable (304), Items (331)

10.21.2 Method overview

Page	Method	Description
331	Add	Add a new string to the hash list.
331	Iterate	Iterate over the strings in the hash table.

10.21.3 Property overview

Page	Properties	Access	Description
331	Items	rw	Key based access to the strings in the hash table.

10.21.4 TFPStringHashTable.Iterate

Synopsis: Iterate over the strings in the hash table.

Declaration:
```
function Iterate(aMethod: TStringIteratorMethod) : AnsiString;  Virtual
function Iterate(aMethod: TStringIteratorCallback) : AnsiString
                ;  Virtual
```

Visibility: public

Description: `Iterate` iterates over all elements in the array, calling `aMethod` for each string, or until the method returns `False` in its `continue` parameter. It returns an empty string if all elements were processed, or the string that was being processed when `aMethod` returned `False` in the `Continue` parameter.

10.21.5 TFPStringHashTable.Add

Synopsis: Add a new string to the hash list.

Declaration:
```
procedure Add(const aKey: AnsiString; const aItem: AnsiString);  Virtual
```

Visibility: public

Description: `Add` adds a new string `AItem` to the hash list with key `AKey`.

Errors: If a string with key `Akey` already exists in the hash table, an exception will be raised.

See also: TFPStringHashTable.Items (331)

10.21.6 TFPStringHashTable.Items

Synopsis: Key based access to the strings in the hash table.

Declaration:
```
Property Items[index: AnsiString]: AnsiString; default
```

Visibility: public

Access: Read,Write

Description: `Items` provides access to the strings in the hash table using their key: the array index `Index` is the key. A key which is not present will result in an empty string.

See also: TFPStringHashTable.Add (331)

10.22 THTCustomNode

10.22.1 Description

`THTCustomNode` is used by the TFPCustomHashTable (298) class to store the keys and associated values.

See also: TFPCustomHashTable (298)

10.22.2 Method overview

Page	Method	Description
332	CreateWith	Create a new instance of `THTCustomNode`.
332	HasKey	Check whether this node matches the given key.

10.22.3 Property overview

Page	Properties	Access	Description
332	Key	r	Key value associated with this hash item.

10.22.4 THTCustomNode.CreateWith

Synopsis: Create a new instance of `THTCustomNode`.

Declaration: `constructor CreateWith(const AString: AnsiString)`

Visibility: public

Description: `CreateWith` creates a new instance of `THTCustomNode` and stores the string `AString` in it. It should never be necessary to call this method directly, it will be called by the TFPCustomHashTable (298) class when needed.

Errors: If no more memory is available, an exception may be raised.

See also: TFPCustomHashTable (298)

10.22.5 THTCustomNode.HasKey

Synopsis: Check whether this node matches the given key.

Declaration: `function HasKey(const AKey: AnsiString) : Boolean`

Visibility: public

Description: `HasKey` checks whether this node matches the given key `AKey`, by comparing it with the stored key. It returns `True` if it does, `False` if not.

Errors: None.

See also: THTCustomNode.Key (332)

10.22.6 THTCustomNode.Key

Synopsis: Key value associated with this hash item.

Declaration: `Property Key : AnsiString`

Visibility: public

Access: Read

Description: `Key` is the key value associated with this hash item. It is stored when the item is created, and is read-only.

See also: THTCustomNode.CreateWith (332)

10.23 THTDataNode

10.23.1 Description

`THTDataNode` is used by TFPDataHashTable (304) to store the hash items in. It simply holds the data pointer.

It should not be necessary to use `THTDataNode` directly, it's only for inner use by `TFPDataHashTable`

See also: TFPDataHashTable (304), THTObjectNode (333), THTStringNode (334)

10.23.2 Property overview

Page	Properties	Access	Description
333	Data	rw	Data pointer.

10.23.3 THTDataNode.Data

Synopsis: Data pointer.

Declaration: `Property Data : pointer`

Visibility: public

Access: Read,Write

Description: Pointer containing the user data associated with the hash value.

10.24 THTObjectNode

10.24.1 Description

`THTObjectNode` is a THTCustomNode (331) descendent which holds the data in the TFPObjectHashTable (320) hash table. It exposes a data string.

It should not be necessary to use `THTObjectNode` directly, it's only for inner use by `TFPObjectHashTable`

See also: TFPObjectHashTable (320)

10.24.2 Property overview

Page	Properties	Access	Description
333	Data	rw	Object instance.

10.24.3 THTObjectNode.Data

Synopsis: Object instance.

Declaration: `Property Data : TObject`

Visibility: public

Access: Read,Write

Description: `Data` is the object instance associated with the key value. It is exposed in TFPObjectHashTable.Items (322)

See also: TFPObjectHashTable (320), TFPObjectHashTable.Items (322), THTOwnedObjectNode (334)

10.25 THTOwnedObjectNode

10.25.1 Description

`THTOwnedObjectNode` is used instead of THTObjectNode (333) in case TFPObjectHashTable (320) owns it's objects. When this object is destroyed, the associated data object is also destroyed.

See also: TFPObjectHashTable (320), THTObjectNode (333), TFPObjectHashTable.OwnsObjects (322)

10.25.2 Method overview

Page	Method	Description
334	Destroy	Destroys the node and the object.

10.25.3 THTOwnedObjectNode.Destroy

Synopsis: Destroys the node and the object.

Declaration: `destructor Destroy;  Override`

Visibility: public

Description: `Destroy` first frees the data object, and then only frees itself.

See also: THTOwnedObjectNode (334), TFPObjectHashTable.OwnsObjects (322)

10.26 THTStringNode

10.26.1 Description

`THTStringNode` is a THTCustomNode (331) descendent which holds the data in the TFPString-HashTable (330) hash table. It exposes a data string.

It should not be necessary to use `THTStringNode` directly, it's only for inner use by `TFPStringHashTable`

See also: TFPStringHashTable (330)

10.26.2 Property overview

Page	Properties	Access	Description
334	Data	rw	String data.

10.26.3 THTStringNode.Data

Synopsis: String data.

Declaration: `Property Data : AnsiString`

Visibility: public

Access: Read,Write

Description: `Data` is the data of this has node. The data is a string, associated with the key. It is also exposed in TFPStringHashTable.Items (331)

See also: TFPStringHashTable (330)

10.27 TObjectBucketList

10.27.1 Description

`TObjectBucketList` is a class that redefines the associative `Data` array using `TObject` instead of `Pointer`. It also adds some overloaded versions of the `Add` and `Remove` calls using `TObject` instead of `Pointer` for the argument and result types.

See also: TObjectBucketList (335)

10.27.2 Method overview

Page	Method	Description
335	Add	Add an object to the list.
335	Remove	Remove an object from the list.

10.27.3 Property overview

Page	Properties	Access	Description
336	Data	rw	Associative array of data items.

10.27.4 TObjectBucketList.Add

Synopsis: Add an object to the list.

Declaration: `function Add(AItem: TObject; AData: TObject) : TObject`

Visibility: public

Description: `Add` adds `AItem` to the list and associated `AData` with it.

See also: TObjectBucketList.Data (336), TObjectBucketList.Remove (335)

10.27.5 TObjectBucketList.Remove

Synopsis: Remove an object from the list.

Declaration: `function Remove(AItem: TObject) : TObject`

Visibility: public

Description: `Remove` removes the object `AItem` from the list. It returns the `Data` object which was associated with the item. If `AItem` was not in the list, then `Nil` is returned.

See also: TObjectBucketList.Add (335), TObjectBucketList.Data (336)

10.27.6 TObjectBucketList.Data

Synopsis: Associative array of data items.

Declaration:
```
Property Data[AItem: TObject]: TObject; default
```

Visibility: public

Access: Read,Write

Description: `Data` provides associative access to the data in the list: it returns the data object associated with the `AItem` object. If the `AItem` object is not in the list, an `EListError` exception is raised.

See also: TObjectBucketList.Add (335)

10.28 TObjectList

10.28.1 Description

`TObjectList` is a TList (**??**) descendent which has as the default array property TObjects (**??**) instead of pointers. By default it also manages the objects: when an object is deleted or removed from the list, it is automatically freed. This behaviour can be disabled when the list is created.

In difference with TFPObjectList (323), `TObjectList` offers a notification mechanism of list change operations: insert, delete. This slows down bulk operations, so if the notifications are not needed, `TFPObjectList` may be more appropriate.

See also: #rtl.classes.TList (**??**), TFPObjectList (323), TComponentList (293), TClassList (290)

10.28.2 Method overview

Page	Method	Description
337	Add	Add an object to the list.
336	Create	Create a new object list.
337	Extract	Extract an object from the list.
338	FindInstanceOf	Search for an instance of a certain class.
338	First	Return the first non-nil object in the list.
338	IndexOf	Search for an object in the list.
338	Insert	Insert an object in the list.
339	Last	Return the last non-nil object in the list.
337	Remove	Remove (and possibly free) an element from the list.

10.28.3 Property overview

Page	Properties	Access	Description
339	Items	rw	Indexed access to the elements of the list.
339	OwnsObjects	rw	Should the list free elements when they are removed.

10.28.4 TObjectList.Create

Synopsis: Create a new object list.

Declaration:
```
constructor Create
constructor Create(FreeObjects: Boolean)
```

Visibility: public

Description: `Create` instantiates a new object list. The `FreeObjects` parameter determines whether objects that are removed from the list should also be freed from memory. By default this is `True`. This behaviour can be changed after the list was instantiated.

Errors: None.

See also: TObjectList.OwnsObjects (339), TFPObjectList (323)

10.28.5 TObjectList.Add

Synopsis: Add an object to the list.

Declaration: `function Add(AObject: TObject) : Integer`

Visibility: public

Description: `Add` overrides the TList (**??**) implementation to accept objects (`AObject`) instead of pointers.

The function returns the index of the position where the object was added.

Errors: If the list must be expanded, and not enough memory is available, an exception may be raised.

See also: TObjectList.Insert (338), #rtl.classes.TList.Delete (**??**), TObjectList.Extract (337), TObjectList.Remove (337)

10.28.6 TObjectList.Extract

Synopsis: Extract an object from the list.

Declaration: `function Extract(Item: TObject) : TObject`

Visibility: public

Description: `Extract` removes the object `Item` from the list if it is present in the list. Contrary to Remove (337), `Extract` does not free the extracted element if OwnsObjects (339) is `True`

The function returns a reference to the item which was removed from the list, or `Nil` if no element was removed.

Errors: None.

See also: TObjectList.Remove (337)

10.28.7 TObjectList.Remove

Synopsis: Remove (and possibly free) an element from the list.

Declaration: `function Remove(AObject: TObject) : Integer`

Visibility: public

Description: `Remove` removes `Item` from the list, if it is present in the list. It frees `Item` if OwnsObjects (339) is `True`, and returns the index of the object that was found in the list, or -1 if the object was not found.

Note that only the first found object is removed from the list.

Errors: None.

See also: TObjectList.Extract (337)

10.28.8 TObjectList.IndexOf

Synopsis: Search for an object in the list.

Declaration:
```
function IndexOf(AObject: TObject) : Integer
```

Visibility: public

Description: `IndexOf` overrides the TList (**??**) implementation to accept an object instance instead of a pointer.
The function returns the index of the first match for `AObject` in the list, or -1 if no match was found.

Errors: None.

See also: TObjectList.FindInstanceOf (338)

10.28.9 TObjectList.FindInstanceOf

Synopsis: Search for an instance of a certain class.

Declaration:
```
function FindInstanceOf(AClass: TClass; AExact: Boolean;
                        AStartAt: Integer) : Integer
```

Visibility: public

Description: `FindInstanceOf` will look through the instances in the list and will return the first instance which is a descendent of class `AClass` if `AExact` is `False`. If `AExact` is true, then the instance should be of class `AClass`.
If no instance of the requested class is found, `Nil` is returned.

Errors: None.

See also: TObjectList.IndexOf (338)

10.28.10 TObjectList.Insert

Synopsis: Insert an object in the list.

Declaration:
```
procedure Insert(Index: Integer; AObject: TObject)
```

Visibility: public

Description: `Insert` inserts `AObject` in the list at position `Index`. The index is zero-based. This method overrides the implementation in TList (**??**) to accept objects instead of pointers.

Errors: If an invalid `Index` is specified, an exception is raised.

See also: TObjectList.Add (337), TObjectList.Remove (337)

10.28.11 TObjectList.First

Synopsis: Return the first non-nil object in the list.

Declaration:
```
function First : TObject
```

Visibility: public

Description: `First` returns a reference to the first non-`Nil` element in the list. If no non-`Nil` element is found, `Nil` is returned.

Errors: None.

See also: TObjectList.Last (339)

10.28.12 TObjectList.Last

Synopsis: Return the last non-nil object in the list.

Declaration: `function Last : TObject`

Visibility: public

Description: `Last` returns a reference to the last non-`Nil` element in the list. If no non-`Nil` element is found, `Nil` is returned.

Errors: None.

See also: TObjectList.First (338)

10.28.13 TObjectList.OwnsObjects

Synopsis: Should the list free elements when they are removed.

Declaration: `Property OwnsObjects : Boolean`

Visibility: public

Access: Read,Write

Description: `OwnsObjects` determines whether the objects in the list should be freed when they are removed (not extracted) from the list, or when the list is cleared. If the property is `True` then they are freed. If the property is `False` the elements are not freed.

The value is usually set in the constructor, and is seldom changed during the lifetime of the list. It defaults to `True`.

See also: TObjectList.Create (336), TObjectList.Remove (337), TObjectList.Extract (337)

10.28.14 TObjectList.Items

Synopsis: Indexed access to the elements of the list.

Declaration: `Property Items[Index: Integer]: TObject; default`

Visibility: public

Access: Read,Write

Description: `Items` is the default property of the list. It provides indexed access to the elements in the list. The index `Index` is zero based, i.e., runs from 0 (zero) to `Count-1`.

See also: #rtl.classes.TList.Count (**??**)

10.29 TObjectQueue

10.29.1 Method overview

Page	Method	Description
340	Peek	Look at the first object in the queue.
340	Pop	Pop the first element off the queue.
340	Push	Push an object on the queue.

10.29.2 TObjectQueue.Push

Synopsis: Push an object on the queue.

Declaration: `function Push(AObject: TObject) : TObject`

Visibility: public

Description: `Push` pushes another object on the queue. It overrides the `Push` method as implemented in `TQueue` so it accepts only objects as arguments.

Errors: If not enough memory is available to expand the queue, an exception may be raised.

See also: TObjectQueue.Pop (340), TObjectQueue.Peek (340)

10.29.3 TObjectQueue.Pop

Synopsis: Pop the first element off the queue.

Declaration: `function Pop : TObject`

Visibility: public

Description: `Pop` removes the first element in the queue, and returns a reference to the instance. If the queue is empty, `Nil` is returned.

Errors: None.

See also: TObjectQueue.Push (340), TObjectQueue.Peek (340)

10.29.4 TObjectQueue.Peek

Synopsis: Look at the first object in the queue.

Declaration: `function Peek : TObject`

Visibility: public

Description: `Peek` returns the first object in the queue, without removing it from the queue. If there are no more objects in the queue, `Nil` is returned.

Errors: None

See also: TObjectQueue.Push (340), TObjectQueue.Pop (340)

10.30 TObjectStack

10.30.1 Description

`TObjectStack` is a stack implementation which manages pointers only.

`TObjectStack` introduces no new behaviour, it simply overrides some methods to accept and/or return `TObject` instances instead of pointers.

See also: TOrderedList (342), TStack (344), TQueue (344), TObjectQueue (339)

10.30.2 Method overview

Page	Method	Description
341	Peek	Look at the top object in the stack.
341	Pop	Pop the top object of the stack.
341	Push	Push an object on the stack.

10.30.3 TObjectStack.Push

Synopsis: Push an object on the stack.

Declaration: `function Push(AObject: TObject) : TObject`

Visibility: public

Description: `Push` pushes another object on the stack. It overrides the `Push` method as implemented in `TStack` so it accepts only objects as arguments.

Errors: If not enough memory is available to expand the stack, an exception may be raised.

See also: TObjectStack.Pop (341), TObjectStack.Peek (341)

10.30.4 TObjectStack.Pop

Synopsis: Pop the top object of the stack.

Declaration: `function Pop : TObject`

Visibility: public

Description: `Pop` pops the top object of the stack, and returns the object instance. If there are no more objects on the stack, `Nil` is returned.

Errors: None

See also: TObjectStack.Push (341), TObjectStack.Peek (341)

10.30.5 TObjectStack.Peek

Synopsis: Look at the top object in the stack.

Declaration: `function Peek : TObject`

Visibility: public

Description: `Peek` returns the top object of the stack, without removing it from the stack. If there are no more objects on the stack, `Nil` is returned.

Errors: None

See also: TObjectStack.Push (341), TObjectStack.Pop (341)

10.31 TOrderedList

10.31.1 Description

`TOrderedList` provides the base class for TQueue (344) and TStack (344). It provides an interface for pushing and popping elements on or off the list, and manages the internal list of pointers.

Note that `TOrderedList` does not manage objects on the stack, i.e. objects are not freed when the ordered list is destroyed.

See also: TQueue (344), TStack (344)

10.31.2 Method overview

Page	Method	Description
343	AtLeast	Check whether the list contains a certain number of elements.
343	Count	Number of elements on the list.
342	Create	Create a new ordered list.
342	Destroy	Free an ordered list.
344	Peek	Return the next element to be popped from the list.
343	Pop	Remove an element from the list.
343	Push	Push another element on the list.

10.31.3 TOrderedList.Create

Synopsis: Create a new ordered list.

Declaration: `constructor Create`

Visibility: public

Description: `Create` instantiates a new ordered list. It initializes the internal pointer list.

Errors: None.

See also: TOrderedList.Destroy (342)

10.31.4 TOrderedList.Destroy

Synopsis: Free an ordered list.

Declaration: `destructor Destroy;  Override`

Visibility: public

Description: `Destroy` cleans up the internal pointer list, and removes the `TOrderedList` instance from memory.

Errors: None.

See also: TOrderedList.Create (342)

10.31.5 TOrderedList.Count

Synopsis: Number of elements on the list.

Declaration: `function Count : Integer`

Visibility: public

Description: `Count` is the number of pointers in the list.

Errors: None.

See also: TOrderedList.AtLeast (343)

10.31.6 TOrderedList.AtLeast

Synopsis: Check whether the list contains a certain number of elements.

Declaration: `function AtLeast(ACount: Integer) : Boolean`

Visibility: public

Description: `AtLeast` returns `True` if the number of elements in the list is equal to or bigger than `ACount`. It returns `False` otherwise.

Errors: None.

See also: TOrderedList.Count (343)

10.31.7 TOrderedList.Push

Synopsis: Push another element on the list.

Declaration: `function Push(AItem: Pointer) : Pointer`

Visibility: public

Description: `Push` adds `AItem` to the list, and returns `AItem`.

Errors: If not enough memory is available to expand the list, an exception may be raised.

See also: TOrderedList.Pop (343), TOrderedList.Peek (344)

10.31.8 TOrderedList.Pop

Synopsis: Remove an element from the list.

Declaration: `function Pop : Pointer`

Visibility: public

Description: `Pop` removes an element from the list, and returns the element that was removed from the list. If no element is on the list, `Nil` is returned.

Errors: None.

See also: TOrderedList.Peek (344), TOrderedList.Push (343)

10.31.9 TOrderedList.Peek

Synopsis: Return the next element to be popped from the list.

Declaration: `function Peek : Pointer`

Visibility: public

Description: `Peek` returns the element that will be popped from the list at the next call to Pop (343), without actually popping it from the list.

Errors: None.

See also: TOrderedList.Pop (343), TOrderedList.Push (343)

10.32 TQueue

10.32.1 Description

`TQueue` is a descendent of TOrderedList (342) which implements Push (343) and Pop (343) behaviour as a queue: what is first pushed on the queue, is popped of first (FIFO: First in, first out).

`TQueue` offers no new methods, it merely implements some abstract methods introduced by TOrderedList (342)

See also: TOrderedList (342), TObjectQueue (339), TStack (344)

10.33 TStack

10.33.1 Description

`TStack` is a descendent of TOrderedList (342) which implements Push (343) and Pop (343) behaviour as a stack: what is last pushed on the stack, is popped of first (LIFO: Last in, first out).

`TStack` offers no new methods, it merely implements some abstract methods introduced by TOrderedList (342)

See also: TOrderedList (342), TObjectStack (340), TQueue (344)

Chapter 11

Reference for unit 'csvdocument'

11.1 Used units

Table 11.1: Used units by unit 'csvdocument'

Name	Page
bufstream	268
Classes	??
Contnrs	284
csvreadwrite	354
System	??
sysutils	??

11.2 Overview

The `CSVDocument` unit offers the TCSVDocument (346) class which can be used to read, manipulate and write the contents of a CSV file. It uses the methods of the #fcl.csvreadwrite.TCSVParser (360) and #fcl.csvreadwrite.TCSVBuilder (355) units to read and write the actual file.

11.3 Constants, types and variables

11.3.1 Types

`TCSVBuilder = csvreadwrite.TCSVBuilder`

`TCSVBuilder` is a backwards-compatibility alias for csvreadwrite.TCSVParser (360)

`TCSVChar = csvreadwrite.TCSVChar`

`TCSVChar` is a backwards-compatibility alias for csvreadwrite.TCSVChar (355)

`TCSVParser = csvreadwrite.TCSVParser`

`TCSVParser` is a backwards-compatibility alias for csvreadwrite.TCSVParser (360)

11.4 TCSVDocument

11.4.1 Description

`TCSVDocument` can be used to read a CSV file in memory using e.g. LoadFromFile (347), manipulate the contents using the Cells (352) property. Additional rows can be added using AddRow (348), additional cells can be added using AddCell (348). After all is done, the SaveToFile (348) method can be used to save the new content to file. The various properties introduced in csvreadwrite.TCSVHandler (345) can be used to configure the format of the CSV file.

See also: LoadFromFile (347), Cells (352), AddRow (348), AddCell (348), SaveToFile (348), csvreadwrite.TCSVHandler (345)

11.4.2 Method overview

Page	Method	Description
348	AddCell	Add a new cell to a row.
348	AddRow	Add a new row to the CSV Data.
351	Clear	Remove all rows.
351	CloneRow	Duplicate a row.
346	Create	Create a new instance of `TCSVDocument`.
347	Destroy	Remove the `TCSVDocument` instance from memory.
351	ExchangeRows	Exchange positions of 2 rows.
350	HasCell	Test if a cell exists.
350	HasRow	Test if a row exists.
350	IndexOfCol	Test whether a value exists at a given row.
350	IndexOfRow	Test whether a value exists at a given column.
349	InsertCell	Insert cell at specified position.
349	InsertRow	Insert row before row number `aRow`.
347	LoadFromFile	Load a CSV file into the document.
347	LoadFromStream	Load CSV data from a stream.
349	RemoveCell	Remove a cell at a particular row.
349	RemoveRow	Remove a row.
351	RemoveTrailingEmptyCells	Remove empty cells at the end of each row.
348	SaveToFile	Save to file on disk.
348	SaveToStream	Save the CSV data to stream.
351	UnifyEmbeddedLineEndings	Ensures all values have the same line ending/.

11.4.3 Property overview

Page	Properties	Access	Description
352	Cells	rw	Array access to all cells.
352	ColCount	r	Return the column count for a given row.
353	CSVText	rw	Return the CSV Document as a single CSV text.
352	DetectBOM	rw	
353	MaxColCount	r	Maximum column count.
352	RowCount	r	Number of available rows.

11.4.4 TCSVDocument.Create

Synopsis: Create a new instance of `TCSVDocument`.

Declaration: `constructor Create;  Override`

Visibility: public

Description: `Create` calls the inherited constructor and then initializes the data structures for the CSV cells.

See also: Cells (352), Destroy (347)

11.4.5 TCSVDocument.Destroy

Synopsis: Remove the `TCSVDocument` instance from memory.

Declaration:
```
destructor Destroy;  Override
```

Visibility: public

Description: `Destroy` cleans up the data structures for the CSV cells and calls the inherited destructor.

See also: Cells (352), Create (346)

11.4.6 TCSVDocument.LoadFromFile

Synopsis: Load a CSV file into the document.

Declaration:
```
procedure LoadFromFile(const AFilename: string);  Overload
procedure LoadFromFile(const AFilename: string; ABufferSize: Integer)
                       ;  Overload
```

Visibility: public

Description: `LoadFromFile` creates a file stream using `aFileName` and calls LoadFromStream (347) to read the contents of the file.

The file is read using an internal buffer for efficiency. The size of the buffer can be specified in bytes using `ABufferSize`. If the size is not specified, a default buffer size is used.

Errors: If the file does not exist, an exception will be raised.

See also: LoadFromStream (347), SaveToStream (348), SaveToFile (348)

11.4.7 TCSVDocument.LoadFromStream

Synopsis: Load CSV data from a stream.

Declaration:
```
procedure LoadFromStream(AStream: TStream)
```

Visibility: public

Description: `LoadFromStream` loads the CSV data from the `aStream` stream. It uses the settings introduced in TCSVHandler (358) when determining fields and rows. If EqualColCountPerRow (360) is `True` then it will add empty cells after reading the CSV data, so all rows have an equal count of columns..

See also: EqualColCountPerRow (360), SaveToStream (348), SaveToFile (348), LoadFromFile (347), TCSVHandler (358)

11.4.8 TCSVDocument.SaveToFile

Synopsis: Save to file on disk.

Declaration: `procedure SaveToFile(const AFilename: string)`

Visibility: public

Description: `SaveToFile` creates a file stream from `aFileName` and calls SaveToStream (348) to actually write the CSV data to the file.

Errors: If the file cannot be created or cannot be written to, an exception will be raised.

See also: SaveToStream (348), LoadFromStream (347), LoadFromFile (347)

11.4.9 TCSVDocument.SaveToStream

Synopsis: Save the CSV data to stream.

Declaration: `procedure SaveToStream(AStream: TStream)`

Visibility: public

Description: `SaveToStream` saves the CSV data to the `aStream` stream. It uses the settings introduced in TCSVHandler (358) to apply the correct formatting to fields and rows. If EqualColCountPerRow (360) is `True` then it will add empty cells prior to writing, so all rows in the file have an equal count of columns.

Errors: If the stream cannot be written to, an exception will be raised.

See also: EqualColCountPerRow (360), SaveToFile (348), LoadFromStream (347), LoadFromFile (347), TCSVHandler (358)

11.4.10 TCSVDocument.AddRow

Synopsis: Add a new row to the CSV Data.

Declaration: `procedure AddRow(const AFirstCell: string)`

Visibility: public

Description: `AddRow` appends a new row to collection of rows, and adds 1 cell to this new row with contents `aFirstCell`. If `aFirstCell` is not specified, then an empty cell is added.

See also: AddCell (348), InsertRow (349), InsertCell (349), RemoveRow (349)

11.4.11 TCSVDocument.AddCell

Synopsis: Add a new cell to a row.

Declaration: `procedure AddCell(ARow: Integer; const AValue: string)`

Visibility: public

Description: `AddCell` adds a new cell at the end of row `aRow` (zero based) with value `aValue`. If `aValue` is not specified, then an empty cell is added.

If a non-existing row is specified, rows are added till `aRow` is reached.

See also: AddRow (348), InsertRow (349), InsertCell (349), RemoveCell (349)

11.4.12 TCSVDocument.InsertRow

Synopsis: Insert row before row number `aRow`.

Declaration: `procedure InsertRow(ARow: Integer; const AFirstCell: string)`

Visibility: public

Description: `InsertRow` inserts a new empty row before row number `aRow`, and adds a cell to the new row with contents `aFirstCell`.

If a non-existing row is specified, the row is simply appended after the last row.

See also: AddRow (348), AddCell (348), InsertCell (349), RemoveRow (349)

11.4.13 TCSVDocument.InsertCell

Synopsis: Insert cell at specified position.

Declaration: `procedure InsertCell(ACol: Integer; ARow: Integer; const AValue: string)`

Visibility: public

Description: `InsertCell` inserts a new cell before cell `aCol` (zero based) in row `aRow` (zero based) with value `aValue`. If `aValue` is not specified, then an empty cell is inserted.

If a non-existing column is specified, the cell is appended at the end of the row.

If a non-existing row is specified, rows are added till `aRow` is reached.

See also: AddRow (348), InsertRow (349), AddCell (348), RemoveCell (349)

11.4.14 TCSVDocument.RemoveRow

Synopsis: Remove a row.

Declaration: `procedure RemoveRow(ARow: Integer)`

Visibility: public

Description: `RemoveRow` removes row number `aRow` (zero-based) from the list of rows. If a non-existing row index is given, no row is removed.

See also: AddRow (348), RemoveCell (349)

11.4.15 TCSVDocument.RemoveCell

Synopsis: Remove a cell at a particular row.

Declaration: `procedure RemoveCell(ACol: Integer; ARow: Integer)`

Visibility: public

Description: `RemoveCell` removes the cell at index `aCol` (zero-based) in row `aRow` (zero-based). If either of `aCol` or `aRow` are invalid, nothing is removed.

See also: AddCell (348), RemoveRow (349), HasCell (350)

11.4.16 TCSVDocument.HasRow

Synopsis: Test if a row exists.

Declaration: `function HasRow(ARow: Integer) : Boolean`

Visibility: public

Description: `HasRow` returns `True` if `aRow` is a valid row index, i.e. is larger than or equal to 0 (zero) and is strictly less than RowCount (352).

See also: RowCount (352), HasCell (350)

11.4.17 TCSVDocument.HasCell

Synopsis: Test if a cell exists.

Declaration: `function HasCell(ACol: Integer; ARow: Integer) : Boolean`

Visibility: public

Description: `HasCell` returns `True` if `aRow` is a valid row index, (i.e. is larger than or equal to 0 (zero) and is strictly less than RowCount (352)) and `aCol` is a valid column index for that row, i.e. is larger than or equal to 0 (zero) and is strictly less than ColCount[aRow] (352)) .

See also: RowCount (352), HasRow (350), ColCount (352)

11.4.18 TCSVDocument.IndexOfCol

Synopsis: Test whether a value exists at a given row.

Declaration: `function IndexOfCol(const AString: string; ARow: Integer) : Integer`

Visibility: public

Description: `IndexOfCol` returns the index of the first cell with given value `aString` in row `aRow`. It returns `-1` if row `aRow` does not exist, or if the value does not appear in the given row.

See also: Cells (352), IndexOfRow (350)

11.4.19 TCSVDocument.IndexOfRow

Synopsis: Test whether a value exists at a given column.

Declaration: `function IndexOfRow(const AString: string; ACol: Integer) : Integer`

Visibility: public

Description: `IndexOfRow` returns the index of the first row with given value `aString` in column `aCol`. It returns `-1` if no such row exists.

See also: IndexOfCol (350), Cells (352)

11.4.20 TCSVDocument.Clear

Synopsis: Remove all rows.

Declaration: `procedure Clear`

Visibility: public

Description: `Clear` removes all rows from the document.

See also: AddRow (348)

11.4.21 TCSVDocument.CloneRow

Synopsis: Duplicate a row.

Declaration: `procedure CloneRow(ARow: Integer; AInsertPos: Integer)`

Visibility: public

Description: `CloneRow` will insert a row at `aInsertPos` and duplicate all cells of row `aRow` in the new row.

See also: AddRow (348), InsertRow (349), ExchangeRows (351)

11.4.22 TCSVDocument.ExchangeRows

Synopsis: Exchange positions of 2 rows.

Declaration: `procedure ExchangeRows(ARow1: Integer; ARow2: Integer)`

Visibility: public

Description: `ExchangeRows` takes 2 rows with positions `aRow1` and `aRow2`, and exchanges them. if either of the row indexes does not exist, no action is performed.

See also: AddRow (348), InsertRow (349), CloneRow (351)

11.4.23 TCSVDocument.UnifyEmbeddedLineEndings

Synopsis: Ensures all values have the same line ending/.

Declaration: `procedure UnifyEmbeddedLineEndings`

Visibility: public

Description: `UnifyEmbeddedLineEndings` forces the line endings in all cell values to match the LineEnding (345) setting. Cell values that do not have a line-endings in them are left untouched.

See also: LineEnding (345), ChangeLineEndings (345)

11.4.24 TCSVDocument.RemoveTrailingEmptyCells

Synopsis: Remove empty cells at the end of each row.

Declaration: `procedure RemoveTrailingEmptyCells`

Visibility: public

Description: `RemoveTrailingEmptyCells` traverses all rows, and removes all empty cells at the end of the row. The first cell of a row is never removed.

See also: RemoveCell (349)

11.4.25 TCSVDocument.DetectBOM

Declaration: `Property DetectBOM : Boolean`

Visibility: public

Access: Read,Write

11.4.26 TCSVDocument.Cells

Synopsis: Array access to all cells.

Declaration: `Property Cells[ACol: Integer;ARow: Integer]: string; default`

Visibility: public

Access: Read,Write

Description: `Cells` provides read and write access to all the cells in the document using `aCol` and `aRow` (both zero-based) as the column and row indexes. If no cell data exists at the location, then reading the value will result in an empty string, and writing will add empty rows and cells so a cell exists at the given position.

See also: AddCell (348), AddRow (348), ColCount (352), RowCount (352)

11.4.27 TCSVDocument.RowCount

Synopsis: Number of available rows.

Declaration: `Property RowCount : Integer`

Visibility: public

Access: Read

Description: `RowCount` returns the number of available rows. Valid row indexes are therefor in the range 0 to `RowCount-1`.

See also: ColCount (352), Cells (352)

11.4.28 TCSVDocument.ColCount

Synopsis: Return the column count for a given row.

Declaration: `Property ColCount[ARow: Integer]: Integer`

Visibility: public

Access: Read

Description: `ColCount` returns the number of available cells in the indicated row (`aRow`, zero-based). Valid column indexes are therefor in the range 0 to `RowCount[aRow]-1`.

See also: RowCount (352), Cells (352)

11.4.29 TCSVDocument.MaxColCount

Synopsis: Maximum column count.

Declaration: `Property MaxColCount : Integer`

Visibility: public

Access: Read

Description: `MaxColCount` scans all rows and returns the largest available ColCount (352) value.

See also: RowCount (352), Cells (352), ColCount (352)

11.4.30 TCSVDocument.CSVText

Synopsis: Return the CSV Document as a single CSV text.

Declaration: `Property CSVText : string`

Visibility: public

Access: Read,Write

Description: `CSVText` calculates the contents of the CSV file as it would be when written using SaveToFile (348) and returns the resulting string.

See also: SaveToFile (348), SaveToStream (348)

Chapter 12

Reference for unit 'csvreadwrite'

12.1 Used units

Table 12.1: Used units by unit 'csvreadwrite'

Name	Page
Classes	??
strutils	??
System	??
sysutils	??

12.2 Overview

The csvreadwrite unit contains a class TCSVParser (360) which allows you to read a CSV file, and TCSVBuilder (355) which allows you to create a CSV file. In both classes, it offers options to handle various kinds of CSV formats. These classes are used in the TCSVDocument (346) class (a class that offers a matrix-like representation of the CVS data) to read and write the CSV file.

12.3 Constants, types and variables

12.3.1 Types

```
TCSVByteOrderMark = (bomNone,bomUTF8,bomUTF16LE,bomUTF16BE)
```

Table 12.2: Enumeration values for type TCSVByteOrderMark

Value	Explanation
bomNone	No BOM marker was read.
bomUTF16BE	The UTF16 big-endian BOM Marker was read.
bomUTF16LE	The UTF16 little-endian BOM Marker was read.
bomUTF8	The UTF8 BOM marker was read.

TCSVByteOrderMark is the type of the TCSVParser.BOM (364) property. It can have the following values

bomNone No BOM marker was read.

bomUTF8 The UTF8 BOM marker was read.

bomUTF16LE The UTF16 little-endian BOM Marker was read.

bomUTF16BE The UTF16 big-endian BOM Marker was read.

```
TCSVChar = Char
```

TCSVChar is an alias type for the basic character used in the CSV file. It is used in TCSVChar (355) and it's descendents to define the character type read from file.

12.4 Procedures and functions

12.4.1 ChangeLineEndings

Synopsis: Change the line endings in a string.

Declaration:
```
function ChangeLineEndings(const AString: string;
                           const ALineEnding: string) : string
```

Visibility: default

Description: ChangeLineEndings is a utility function which changes the line ending characters CR, CR/LF and LF in Astring to the specified line ending aLineEnding. It returns the transformed string.

12.5 TCSVBuilder

12.5.1 Description

TCSVBuilder can be used to correctly write a CSV file. To use it, optionally set a stream to which the CSV file will be written using SetOutput (356), call AppendCell (357) for each cell you wish to write, and call AppendRow (357) when you want to start a new line. If you didn't specify an output stream, the output is available in DefaultOutput (357) or DefaultOutputAsString (357).

See also: SetOutput (356), AppendCell (357), AppendRow (357), DefaultOutput (357), DefaultOutputAsString (357)

12.5.2 Method overview

Page	Method	Description
357	AppendCell	Append a cell to the output.
357	AppendRow	Append a new row.
356	Create	Create a new instance of TCSVBuilder.
356	Destroy	Free a TCSVBuilder instance.
356	ResetBuilder	Reset values.
356	SetOutput	Set the output stream to write to.

12.5.3 Property overview

Page	Properties	Access	Description
357	DefaultOutput	r	Default output as stream.
357	DefaultOutputAsString	r	Default output as string value.

12.5.4 TCSVBuilder.Create

Synopsis: Create a new instance of `TCSVBuilder`.

Declaration:
```
constructor Create;  Override
```

Visibility: public

Description: `Create` calls the inherited constructor and initializes the default output.

See also: DefaultOutput (357), DefaultOutputAsString (357)

12.5.5 TCSVBuilder.Destroy

Synopsis: Free a `TCSVBuilder` instance.

Declaration:
```
destructor Destroy;  Override
```

Visibility: public

Description: `Destroy` cleans up the default output and calls the inherited constructor.

See also: DefaultOutput (357), DefaultOutputAsString (357)

12.5.6 TCSVBuilder.SetOutput

Synopsis: Set the output stream to write to.

Declaration:
```
procedure SetOutput(AStream: TStream)
```

Visibility: public

Description: `SetOutput` can be used to set the output stream to `aStream`. The use of this is optional, as the output is available by default in DefaultOutput (357) or DefaultOutputAsString (357). Calling `SetOutpu` will result in resetting the inner state of the builder using ResetBuilder (356) , so it must not be called if you already wrote some data.

See also: DefaultOutput (357), DefaultOutputAsString (357), ResetBuilder (356)

12.5.7 TCSVBuilder.ResetBuilder

Synopsis: Reset values.

Declaration:
```
procedure ResetBuilder
```

Visibility: public

Description: `ResetBuilder` resets the builder state to the initial state, at the beginning of the stream. It is called when SetOutput (356) is called.

See also: SetOutput (356)

12.5.8 TCSVBuilder.AppendCell

Synopsis: Append a cell to the output.

Declaration: `procedure AppendCell(const AValue: string)`

Visibility: public

Description: `AppendCell` appends a cell with content `AValue` to the output; It will quote the value using QuoteChar (359) if necessary, and append the necessary Delimiter (359) characters. It will not move to the next line, for this AppendRow (357) must be called.

See also: QuoteChar (359), Delimiter (359), AppendRow (357)

12.5.9 TCSVBuilder.AppendRow

Synopsis: Append a new row.

Declaration: `procedure AppendRow`

Visibility: public

Description: `AppendRow` moves the cell pointer to the new row, i.e. it appends a LineEnding (359) character.

See also: LineENding (359), AppendCell (357), EqualColCountPerRow (360)

12.5.10 TCSVBuilder.DefaultOutput

Synopsis: Default output as stream.

Declaration: `Property DefaultOutput : TMemoryStream`

Visibility: public

Access: Read

Description: `DefaultOutput` is a stream to which the output is written if no output stream was specified with the SetOutput (356) call.

See also: SetOutput (356), DefaultOutputAsString (357)

12.5.11 TCSVBuilder.DefaultOutputAsString

Synopsis: Default output as string value.

Declaration: `Property DefaultOutputAsString : string`

Visibility: public

Access: Read

Description: `DefaultOutput` is a string to which the output is written if no output stream was specified with the SetOutput (356) call. It is basically the contents of the DefaultOutput (357) property as a string.

See also: SetOutput (356), DefaultOutput (357)

12.6 TCSVHandler

12.6.1 Description

`TCSVHandler` is the base class for class TCSVParser (360) which allows you to read a CSV file, and class TCSVBuilder (355) which allows you to write a CSV file. It defines some common properties for these classes, mainly to describe the formatting of the CSV file, such as the Delimiter (359) or QuoteChar (359) and LineEnding (359) properties.

Normally you will not create an instance of `TCSVHandler`, instead you will create a `TCSVParser` or `TCSVBuilder` instance.

See also: TCSVParser (360), TCSVBuilder (355), Delimiter (359), QuoteChar (359), LineEnding (359)

12.6.2 Method overview

Page	Method	Description
358	Assign	Assign properties from another `TCSVHandler` instance.
359	AssignCSVProperties	Assign properties from another `TCSVHandler` instance.
358	Create	Create an instance of `TCSVHandler`.

12.6.3 Property overview

Page	Properties	Access	Description
359	Delimiter	rw	Field delimiter character.
360	EqualColCountPerRow	rw	Ensure every row has an equal amount of columns.
360	IgnoreOuterWhitespace	rw	Ignore whitespace between delimiters and field data.
359	LineEnding	rw	Line ending character.
359	QuoteChar	rw	Character to quote values.
360	QuoteOuterWhitespace	rw	Write quotes when outer whitespace is found in a value.

12.6.4 TCSVHandler.Create

Synopsis: Create an instance of `TCSVHandler`.

Declaration:
```
constructor Create;  Virtual
```

Visibility: public

Description: `Create` calls the inherited constructor and initializes various properties such as Delimiter (359), QuoteChar (359) and LineEnding (359) to their initial values.

See also: Delimiter (359), QuoteChar (359), LineEnding (359)

12.6.5 TCSVHandler.Assign

Synopsis: Assign properties from another `TCSVHandler` instance.

Declaration:
```
procedure Assign(ASource: TPersistent);  Override
```

Visibility: public

Description: `Assign` overrides TPersistent.Assign (**??**) to copy all `TCSVHandler` properties from the `aSource` instance to the current instance. It calls AssignCSVProperties (359) to do the actual copying.

See also: TPersistent.Assign (**??**)

12.6.6 TCSVHandler.AssignCSVProperties

Synopsis: Assign properties from another `TCSVHandler` instance.

Declaration: `procedure AssignCSVProperties(ASource: TCSVHandler)`

Visibility: public

Description: `AssignCSVProperties` is called by Assign (358) to copy all `TCSVHandler` properties from the `aSource` instance to the current instance.

See also: Assign (358)

12.6.7 TCSVHandler.Delimiter

Synopsis: Field delimiter character.

Declaration: `Property Delimiter : TCSVChar`

Visibility: public

Access: Read,Write

Description: `Delimiter` is the field delimiter character. By default, it is the comma (`,`). Values that contain a delimiter character must be quoted by the character specified in QuoteChar (359).

See also: QuoteChar (359), LineEnding (359)

12.6.8 TCSVHandler.QuoteChar

Synopsis: Character to quote values.

Declaration: `Property QuoteChar : TCSVChar`

Visibility: public

Access: Read,Write

Description: `QuoteChar` specifies the character to use when quoting values. Between the quote characters, field delimiter (as set in Delimiter (359)) or line ending (as set in LineEnding (359)) characters lose their special meaning and are considered part of the value. The default quote character is the double quote (`"`).

See also: Delimiter (359), LineEnding (359)

12.6.9 TCSVHandler.LineEnding

Synopsis: Line ending character.

Declaration: `Property LineEnding : string`

Visibility: public

Access: Read,Write

Description: `LineEnding` specifies the line-ending character. It is initialized with the current system's line ending character - one of CR, CR/LF or LF.

12.6.10 TCSVHandler.IgnoreOuterWhitespace

Synopsis: Ignore whitespace between delimiters and field data.

Declaration: `Property IgnoreOuterWhitespace : Boolean`

Visibility: public

Access: Read,Write

Description: `IgnoreOuterWhitespace` can be set to `True` to let the parser ignore any whitespace between the value of a field and the delimiter characters Delimiter (359) when reading data. The default is `False`

See also: Delimiter (359), QuoteOuterWhitespace (360)

12.6.11 TCSVHandler.QuoteOuterWhitespace

Synopsis: Write quotes when outer whitespace is found in a value.

Declaration: `Property QuoteOuterWhitespace : Boolean`

Visibility: public

Access: Read,Write

Description: `QuoteOuterWhitespace` can be set to `True` to let the parser quote values that have whitespace at the beginning or the end of the value. If set to `False`, there will be whitespace between the value of a field and the delimiter characters Delimiter (359), which can alter the value when reading data depending on the setting of IgnoreOuterWhitespace (360). The default is `True`.

See also: Delimiter (359), IgnoreOuterWhitespace (360)

12.6.12 TCSVHandler.EqualColCountPerRow

Synopsis: Ensure every row has an equal amount of columns.

Declaration: `Property EqualColCountPerRow : Boolean`

Visibility: public

Access: Read,Write

Description: `EqualColCountPerRow` must be set to report an equal amount of columns for every row when reading and writing. Empty columns will be reported or added for every row where the number of columns is less than the maximal amount of columns. This property is not used in the TCSVParser (360) or TCSVBuilder (355) classes. It is used in the TCSVDocument (354) class. The default is `True`.

See also: TCSVParser (360), TCSVBuilder (355), TCSVDocument (354)

12.7 TCSVParser

12.7.1 Description

`TCSVParser` can be used to read a CSV file. To use it, the stream or string containing the CSV data must be specified with SetSource (362), after which ParseNextCell (362) can be called till it returns

false. The current cell value after `ParseNextCell` returns `True` is available in CurrentCellText (363). By tracking CurrentRow (362), CurrentCol (363) and MaxColCount (363), the structure of the CSV file can be determined.

See also: SetSource (362), ParseNextCell (362), CurrentCellText (363), CurrentRow (362), CurrentCol (363), MaxColCount (363)

12.7.2 Method overview

Page	Method	Description
361	Create	Create a new instance of `TCSVParser`.
361	Destroy	Free the `TCSVParser` instance.
362	ParseNextCell	Parse the next cell.
362	ResetParser	Reset the parser to its initial state.
362	SetSource	Set the CSV source data.

12.7.3 Property overview

Page	Properties	Access	Description
364	BOM	r	type of BOM marker found at the start of the data.
363	CurrentCellText	r	Current field value.
363	CurrentCol	r	Column (zero based) of the current field.
362	CurrentRow	r	Row (zero based) of the current field.
364	DetectBOM	rw	Must the parser attempt to read the BOM marker ?
364	FreeStream	rw	Does the parser free the stream when done ?
363	MaxColCount	r	Return the maximum column count encountered till now.

12.7.4 TCSVParser.Create

Synopsis: Create a new instance of `TCSVParser`.

Declaration: `constructor Create;  Override`

Visibility: public

Description: `TCSVParser` calls the inherited constructor and initializes some internal structures.

See also: TCSVHandler.Create (358)

12.7.5 TCSVParser.Destroy

Synopsis: Free the `TCSVParser` instance.

Declaration: `destructor Destroy;  Override`

Visibility: public

Description: `TCSVParser` clears the internal data and calls the inherited destructor.

See also: TCSVParser.Create (361)

12.7.6 TCSVParser.SetSource

Synopsis: Set the CSV source data.

Declaration:
```
procedure SetSource(AStream: TStream);  Overload
procedure SetSource(const AString: string);  Overload
```

Visibility: public

Description: `SetSource` sets the source of the CSV data to `aStream` (a stream) or `aString` (a string). It calls ResetParser (362) to reset the parser state. The stream is by default not owned by the parser, i.e. you must free it after it has been used. If TCSVParser.FreeStream (364) is `True`, then the stream will be freed by the parser class when it is destroyed, or when a new stream is set.

See also: FreeStream (364), ResetParser (362)

12.7.7 TCSVParser.ResetParser

Synopsis: Reset the parser to its initial state.

Declaration:
```
procedure ResetParser
```

Visibility: public

Description: `ResetParser` resets the parser to its initial state: CurrentRow (362), CurrentCol (363) and MaxColCount (363) are all set to zero, and the output is cleared. The stream is put at position zero and if DetectBOM (364) is `True`, the BOM (364) marker is read.

See also: SetSource (362), CurrentRow (362), CurrentCol (363), MaxColCount (363), BOM (364)

12.7.8 TCSVParser.ParseNextCell

Synopsis: Parse the next cell.

Declaration:
```
function ParseNextCell : Boolean
```

Visibility: public

Description: `ParseNextCell` attempts to read the next field in the CSV data, moving to the next row if necessary. It returns `True` if a cell was read successfully, `False` if no more CSV data is available - when the stream has reached EOF. It takes into account the Delimiter (359), QuoteChar (359) and LineEnding (359) properties to determine the field boundaries. When it has returned `True`, the properties CurrentRow (362), CurrentCol (363), MaxColCount (363) and CurrentCellText (363) can be used to determine what field was read and what the contents of the field were.

See also: CurrentRow (362), CurrentCol (363), MaxColCount (363), CurrentCellText (363), Delimiter (359), QuoteChar (359), LineEnding (359)

12.7.9 TCSVParser.CurrentRow

Synopsis: Row (zero based) of the current field.

Declaration:
```
Property CurrentRow : Integer
```

Visibility: public

Access: Read

Description: `CurrentRow` contains the row number (zero based) of the current field (cell) in the CSV file. This value is only valid after `ParseNextCell` returns `True`.

See also: ParseNextCell (362), CurrentCol (363), MaxColCount (363), CurrentCellText (363)

12.7.10 TCSVParser.CurrentCol

Synopsis: Column (zero based) of the current field.

Declaration: `Property CurrentCol : Integer`

Visibility: public

Access: Read

Description: `CurrentCol` contains the column number (zero based) of the current field (cell) in the CSV file. This value is only valid after `ParseNextCell` returns `True`.

See also: CurrentRow (362), ParseNextCell (362), MaxColCount (363), CurrentCellText (363)

12.7.11 TCSVParser.CurrentCellText

Synopsis: Current field value.

Declaration: `Property CurrentCellText : string`

Visibility: public

Access: Read

Description: `CurrentCellText` contains the value of the current field (cell) in the CSV file. This value is only valid after `ParseNextCell` returns `True`. The value has already been processed according to the QuoteOuterWhitespace (360) property.

See also: CurrentRow (362), CurrentCol (363), MaxColCount (363), ParseNextCell (362), QuoteOuterWhitespace (360)

12.7.12 TCSVParser.MaxColCount

Synopsis: Return the maximum column count encountered till now.

Declaration: `Property MaxColCount : Integer`

Visibility: public

Access: Read

Description: `MaxColCount` contains the maximum column count encoutered till now. This value will be updated as ParseNextCell (362) is called, and consequently the final value is only available after `ParseNextValue` returned `False`.

See also: CurrentRow (362), CurrentCol (363), CurrentCellText (363), ParseNextCell (362)

12.7.13 TCSVParser.FreeStream

Synopsis: Does the parser free the stream when done ?

Declaration: `Property FreeStream : Boolean`

Visibility: public

Access: Read,Write

Description: `FreeStream` determines whether the parser frees the stream when done or not. The stream is by default not owned by the parser, i.e. you must free it after it has been used. If `FreeStream`is `True`, then the stream will be freed by the parser class when it is destroyed, or when a new stream is set using SetSource (362)

See also: SetSource (362)

12.7.14 TCSVParser.BOM

Synopsis: type of BOM marker found at the start of the data.

Declaration: `Property BOM : TCSVByteOrderMark`

Visibility: public

Access: Read

Description: `BOM` indicates what Byte Order Marker was found at the beginning of the data. The value is updated as soon as the CSV is set using TCSVParser.SetSource (362) and DetectBOM (364) is `True`. For a list of possible values, see TCSVByteOrderMark (354).

See also: TCSVByteOrderMark (354), TCSVParser.SetSource (362)

12.7.15 TCSVParser.DetectBOM

Synopsis: Must the parser attempt to read the BOM marker ?

Declaration: `Property DetectBOM : Boolean`

Visibility: public

Access: Read,Write

Description: `DetectBOM` can be set to `True` if you want the CSV parser to attempt to detect a BOM marker. If set to `True`, then ResetParser (362) will attempt to read the BOM marker when the CSV data is set using SetSource (362) The result of the detection is available the in BOM (364) property) after the source data is set. It follows that `DetectBOM` must be set before calling SetSource (362).

See also: SetSource (362), ResetParser (362), BOM (364)

Chapter 13

Reference for unit 'CustApp'

13.1 Used units

Table 13.1: Used units by unit 'CustApp'

Name	Page
Classes	??
singleinstance	1481
System	??
sysutils	??

13.2 Overview

The CustApp unit implements the TCustomApplication (366) class, which serves as the common ancestor to many kinds of `TApplication` classes: a GUI application in the LCL, a CGI application in FPCGI, a daemon application in daemonapp. It introduces some properties to describe the environment in which the application is running (environment variables, program command-line parameters) and introduces some methods to initialize and run a program, as well as functionality to handle exceptions.

Typical use of a descendent class is to introduce a global variable `Application` and use the following code:

```
Application.Initialize;
  Application.Run;
```

Since normally only a single instance of this class is created, and it is a `TComponent` descendent, it can be used as an owner for many components, doing so will ensure these components will be freed when the application terminates.

13.3 Constants, types and variables

13.3.1 Types

```
TEventLogTypes = Set of TEventType
```

TEventLogTypes is a set of TEventType (**??**), used in TCustomApplication.EventLogFilter (378) to filter events that are sent to the system log.

```
TExceptionEvent = procedure(Sender: TObject; E: Exception) of
  object
```

TExceptionEvent is the prototype for the exception handling events in TCustomApplication.

```
TStringArray = Array of string
```

TStringArray is an array of strings, used in the TCustomApplication.GetOptionValues (371) call.

13.3.2 Variables

```
CustomApplication : TCustomApplication = Nil
```

CustomApplication contains the global application instance. All descendents of TCustomApplication (366) should, in addition to storing an instance pointer in some variable (most likely called "Application") store the instance pointer in this variable. This ensures that, whatever kind of application is being created, user code can access the application object.

13.4 TCustomApplication

13.4.1 Description

TCustomApplication is the ancestor class for classes that wish to implement a global application class instance. It introduces several application-wide functionalities.

- Exception handling in HandleException (368), ShowException (369), OnException (374) and StopOnException (377).
- Command-line parameter parsing in FindOptionIndex (370), GetOptionValue (370), CheckOptions (371) and HasOption (371)
- Environment variable handling in GetEnvironmentList (373) and EnvironmentVariable (376).

Descendent classes need to override the DoRun protected method to implement the functionality of the program.

13.4.2 Method overview

Page	Method	Description
371	CheckOptions	Check whether all given options on the command-line are valid.
367	Create	Create a new instance of the `TCustomApplication` class.
368	Destroy	Destroys the `TCustomApplication` instance.
370	FindOptionIndex	Return the index of an option.
373	GetEnvironmentList	Return a list of environment variables.
372	GetNonOptions	Get all non-switch options.
370	GetOptionValue	Return the value of a command-line option.
371	GetOptionValues	Get the values for an option that may be specified multiple times.
368	HandleException	Handle an exception.
371	HasOption	Check whether an option was specified.
368	Initialize	Initialize the application.
373	Log	Write a message to the event log.
369	Run	Runs the application.
369	ShowException	Show an exception to the user.
369	Terminate	Terminate the application.

13.4.3 Property overview

Page	Properties	Access	Description
377	CaseSensitiveLongOptions	rw	
376	CaseSensitiveOptions	rw	Are options interpreted case sensitive or not.
377	CaseSensitiveShortOptions	rw	
375	ConsoleApplication	r	Is the application a console application or not.
376	EnvironmentVariable	r	Environment variable access.
378	EventLogFilter	rw	Event to filter events, before they are sent to the system log.
377	ExceptionExitCode	rw	ExitCode to use then terminating the program due to an exception.
373	ExeName	r	Name of the executable.
374	HelpFile	rw	Location of the application help file.
375	Location	r	Application location.
374	OnException	rw	Exception handling event.
376	OptionChar	rw	Command-line switch character.
376	ParamCount	r	Number of command-line parameters.
375	Params	r	Command-line parameters.
378	SingleInstance	r	Single instance used to control single application instance behaviour.
378	SingleInstanceClass	rw	Class to use when creating single instance.
378	SingleInstanceEnabled	rw	Enable single application instance control.
377	StopOnException	rw	Should the program loop stop on an exception.
374	Terminated	r	Was `Terminate` called or not.
374	Title	rw	Application title.

13.4.4 TCustomApplication.Create

Synopsis: Create a new instance of the `TCustomApplication` class.

Declaration: `constructor Create(AOwner: TComponent); Override`

Visibility: public

Description: `Create` creates a new instance of the `TCustomApplication` class. It sets some defaults for the various properties, and then calls the inherited `Create`.

See also: TCustomApplication.Destroy (368)

13.4.5 TCustomApplication.Destroy

Synopsis: Destroys the `TCustomApplication` instance.

Declaration: `destructor Destroy;  Override`

Visibility: public

Description: `Destroy` simply calls the inherited `Destroy`.

See also: TCustomApplication.Create (367)

13.4.6 TCustomApplication.HandleException

Synopsis: Handle an exception.

Declaration: `procedure HandleException(Sender: TObject);  Virtual`

Visibility: public

Description: `HandleException` is called (or can be called) to handle the exception `Sender`. If the exception is not of class `Exception` then the default handling of exceptions in the SysUtils unit is called.

If the exception is of class `Exception` and the OnException (374) handler is set, the handler is called with the exception object and `Sender` argument.

If the `OnException` handler is not set, then the exception is passed to the ShowException (369) routine, which can be overridden by descendent application classes to show the exception in a way that is fit for the particular class of application. (a GUI application might show the exception in a message dialog.

When the exception is handled in the above manner, and the StopOnException (377) property is set to `True`, the Terminated (374) property is set to `True`, which will cause the Run (369) loop to stop, and the application will exit.

See also: ShowException (369), StopOnException (377), Terminated (374), Run (369)

13.4.7 TCustomApplication.Initialize

Synopsis: Initialize the application.

Declaration: `procedure Initialize;  Virtual`

Visibility: public

Description: `Initialize` can be overridden by descendent applications to perform any initialization after the class was created. It can be used to react to properties being set at program startup. End-user code should call `Initialize` prior to calling `Run`

In `TCustomApplication`, `Initialize` sets `Terminated` to `False`.

See also: TCustomApplication.Run (369), TCustomApplication.Terminated (374)

13.4.8 TCustomApplication.Run

Synopsis: Runs the application.

Declaration:
```
procedure Run
```

Visibility: public

Description: `Run` is the start of the user code: when called, it starts a loop and repeatedly calls `DoRun` until `Terminated` is set to `True`. If an exception is raised during the execution of `DoRun`, it is caught and handled to TCustomApplication.HandleException (368). If TCustomApplication.StopOnException (377) is set to `True` (which is *not* the default), `Run` will exit, and the application will then terminate. The default is to call `DoRun` again, which is useful for applications running a message loop such as services and GUI applications.

See also: TCustomApplication.HandleException (368), TCustomApplication.StopOnException (377)

13.4.9 TCustomApplication.ShowException

Synopsis: Show an exception to the user.

Declaration:
```
procedure ShowException(E: Exception);  Virtual
```

Visibility: public

Description: `ShowException` should be overridden by descendent classes to show an exception message to the user. The default behaviour is to call the ShowException (**??**) procedure in the SysUtils unit.

Descendent classes should do something appropriate for their context: GUI applications can show a message box, daemon applications can write the exception message to the system log, web applications can send a 500 error response code.

Errors: None.

See also: ShowException (**??**), TCustomApplication.HandleException (368), TCustomApplication.StopOnException (377)

13.4.10 TCustomApplication.Terminate

Synopsis: Terminate the application.

Declaration:
```
procedure Terminate;  Virtual
procedure Terminate(AExitCode: Integer);  Virtual
```

Visibility: public

Description: `Terminate` sets the `Terminated` property to `True`. By itself, this does not terminate the application. Instead, descendent classes should in their `DoRun` method, check the value of the Terminated (374) property and properly shut down the application if it is set to `True`.

When `AExitCode` is specified, it will passed to System.ExitCode (365), and when the program is halted, that is the exit code of the program as returned to the OS. If the application is terminated due to an exception, ExceptionExitCode (377) will be used as the value for this argument.

See also: TCustomApplication.Terminated (374), TCustomApplication.Run (369), ExceptionExitCode (377), System.ExitCode (365)

13.4.11 TCustomApplication.FindOptionIndex

Synopsis: Return the index of an option.

Declaration:
```
function FindOptionIndex(const S: string; var Longopt: Boolean;
                         StartAt: Integer) : Integer
```

Visibility: public

Description: `FindOptionIndex` will return the index of the option `S` or the long option `LongOpt`. Neither of them should include the switch character. If no such option was specified, -1 is returned. If either the long or short option was specified, then the position on the command-line is returned.

Depending on the value of the CaseSensitiveOptions (376) property, the search is performed case sensitive or case insensitive.

Options are identified as command-line parameters which start with OptionChar (376) (by default the dash ('-') character).

See also: HasOption (371), GetOptionValue (370), CheckOptions (371), CaseSensitiveOptions (376), OptionChar (376)

13.4.12 TCustomApplication.GetOptionValue

Synopsis: Return the value of a command-line option.

Declaration:
```
function GetOptionValue(const S: string) : string
function GetOptionValue(const C: Char; const S: string) : string
```

Visibility: public

Description: `GetOptionValue` returns the value of an option. Values are specified in the usual GNU option format, either of

```
--longopt=Value
```

or

```
-c Value
```

is supported.

The function returns the specified value, or the empty string if none was specified.

Depending on the value of the CaseSensitiveOptions (376) property, the search is performed case sensitive or case insensitive.

Options are identified as command-line parameters which start with OptionChar (376) (by default the dash ('-') character).

If an option can appear multiple times, use TCustomApplication.GetOptionValues (371) to retrieve all values. This function only returns the value of the first occurrence of an option.

See also: FindOptionIndex (370), HasOption (371), CheckOptions (371), CaseSensitiveOptions (376), OptionChar (376), TCustomApplication.GetOptionValues (371)

13.4.13 TCustomApplication.GetOptionValues

Synopsis: Get the values for an option that may be specified multiple times.

Declaration:
```
function GetOptionValues(const C: Char; const S: string) : TStringArray
```

Visibility: public

Description: `GetOptionValues` returns all values specified by command-line option switches `C` or `S`. For each occurrence of the command-line option `C` or `S`, the associated value is added to the array.

TCustomApplication.GetOptionValue (370) will only return the first occurrence of a value.

Errors: None.

See also: TCustomApplication.GetOptionValue (370)

13.4.14 TCustomApplication.HasOption

Synopsis: Check whether an option was specified.

Declaration:
```
function HasOption(const S: string) : Boolean
function HasOption(const C: Char; const S: string) : Boolean
```

Visibility: public

Description: `HasOption` returns `True` if the specified option was given on the command line. Either the short option character `C` or the long option `S` may be used. Note that both options (requiring a value) and switches can be specified.

Depending on the value of the CaseSensitiveOptions (376) property, the search is performed case sensitive or case insensitive.

Options are identified as command-line parameters which start with OptionChar (376) (by default the dash ('-') character).

See also: FindOptionIndex (370), GetOptionValue (370), CheckOptions (371), CaseSensitiveOptions (376), OptionChar (376)

13.4.15 TCustomApplication.CheckOptions

Synopsis: Check whether all given options on the command-line are valid.

Declaration:
```
function CheckOptions(const ShortOptions: string;
                      const Longopts: TStrings; Opts: TStrings;
                      NonOpts: TStrings; AllErrors: Boolean) : string
function CheckOptions(const ShortOptions: string;
                      const Longopts: Array of string; Opts: TStrings;
                      NonOpts: TStrings; AllErrors: Boolean) : string
function CheckOptions(const ShortOptions: string;
                      const Longopts: TStrings; AllErrors: Boolean)
                       : string
function CheckOptions(const ShortOptions: string;
                      const LongOpts: Array of string;
                      AllErrors: Boolean) : string
function CheckOptions(const ShortOptions: string;
                      const LongOpts: string; AllErrors: Boolean)
                       : string
```

Visibility: public

Description: `CheckOptions` scans the command-line and checks whether the options given are valid options. It also checks whether options that require a valued are indeed specified with a value.

The `ShortOptions` contains a string with valid short option characters. Each character in the string is a valid option character. If a character is followed by a colon (:), then a value must be specified. If it is followed by 2 colon characters (::) then the value is optional.

`LongOpts` is a list of strings (which can be specified as an array, a `TStrings` instance or a string with whitespace-separated values) of valid long options.

When the function returns, if `Opts` is non-`Nil`, the `Opts` stringlist is filled with the passed valid options. If `NonOpts` is non-nil, it is filled with any non-option strings that were passed on the command-line.

The function returns an empty string if all specified options were valid options, and whether options requiring a value have a value. If an error was found during the check, the return value is a string describing the error.

Options are identified as command-line parameters which start with OptionChar (376) (by default the dash ('-') character).

if `AllErrors` is `True` then all errors are returned, separated by a sLineBreak (**??**) character.

Errors: If an error was found during the check, the return value is a string describing the error(s).

See also: FindOptionIndex (370), GetOptionValue (370), HasOption (371), CaseSensitiveOptions (376), OptionChar (376)

13.4.16 TCustomApplication.GetNonOptions

Synopsis: Get all non-switch options.

Declaration:
```
function GetNonOptions(const ShortOptions: string;
                       const Longopts: Array of string) : TStringArray
procedure GetNonOptions(const ShortOptions: string;
                        const Longopts: Array of string;
                        NonOptions: TStrings)
```

Visibility: public

Description: `GetNonOptions` returns the items on the command-line that are not associated with a switch. It checks the command-line for allowed switches as they are indicated by `ShortOptions` and `Longopts`. The format is identical to TCustomApplication.Checkoptions (371). This is useful for an application which accepts a command form such as `svn`:

```
svn commit [options] files
```

In the above example, "commit" and "files" would be returned by `GetNonOptions`

The non-options are returned in the form of a string array, or a stringlist instance can be passed in `NonOptions`. Either will be filled with the non-options on return.

Errors: None.

See also: TCustomApplication.HasOption (371), TCustomApplication.Checkoptions (371), TCustomApplication.GetOptionValue (370), TCustomApplication.GetOptionValues (371)

13.4.17 TCustomApplication.GetEnvironmentList

Synopsis: Return a list of environment variables.

Declaration:
```
procedure GetEnvironmentList(List: TStrings; NamesOnly: Boolean)
procedure GetEnvironmentList(List: TStrings)
```

Visibility: public

Description: `GetEnvironmentList` returns a list of environment variables in `List`. They are in the form `Name=Value`, one per item in `list`. If `NamesOnly` is `True`, then only the names are returned.

See also: EnvironmentVariable (376)

13.4.18 TCustomApplication.Log

Synopsis: Write a message to the event log.

Declaration:
```
procedure Log(EventType: TEventType; const Msg: string)
procedure Log(EventType: TEventType; const Fmt: string;
              const Args: Array of const)
```

Visibility: public

Description: `Log` is meant for all applications to have a default logging mechanism. By default it does not do anything, descendent classes should override this method to provide appropriate logging: they should write the message `Msg` with type `EventType` to some log mechanism such as #fcl.eventlog.TEventLog (660)

The second form using `Fmt` and `Args` will format the message using the provided arguments prior to logging it.

Errors: None.

See also: #rtl.sysutils.TEventType (**??**)

13.4.19 TCustomApplication.ExeName

Synopsis: Name of the executable.

Declaration:
```
Property ExeName : string
```

Visibility: public

Access: Read

Description: `ExeName` returns the full name of the executable binary (path+filename). This is equivalent to `Paramstr(0)`

Note that some operating systems do not return the full pathname of the binary.

See also: ParamStr (**??**)

13.4.20 TCustomApplication.HelpFile

Synopsis: Location of the application help file.

Declaration: `Property HelpFile : string`

Visibility: public

Access: Read,Write

Description: `HelpFile` is the location of the application help file. It is a simple string property which can be set by an IDE such as Lazarus, and is mainly provided for compatibility with Delphi's `TApplication` implementation.

See also: TCustomApplication.Title (374)

13.4.21 TCustomApplication.Terminated

Synopsis: Was `Terminate` called or not.

Declaration: `Property Terminated : Boolean`

Visibility: public

Access: Read

Description: `Terminated` indicates whether Terminate (369) was called or not. Descendent classes should check `Terminated` at regular intervals in their implementation of `DoRun`, and if it is set to `True`, should exit gracefully the `DoRun` method.

See also: Terminate (369)

13.4.22 TCustomApplication.Title

Synopsis: Application title.

Declaration: `Property Title : string`

Visibility: public

Access: Read,Write

Description: `Title` is a simple string property which can be set to any string describing the application. It does nothing by itself, and is mainly introduced for compatibility with Delphi's `TApplication` implementation.

See also: HelpFile (374)

13.4.23 TCustomApplication.OnException

Synopsis: Exception handling event.

Declaration: `Property OnException : TExceptionEvent`

Visibility: public

Access: Read,Write

Description: `OnException` can be set to provide custom handling of exceptions, instead of the default action, which is simply to show the exception using ShowException (369).

If the event is set, then it is called by the HandleException (368) routine. Do not use the `OnException` event directly, instead call `HandleException`.

See also: ShowException (369)

13.4.24 TCustomApplication.ConsoleApplication

Synopsis: Is the application a console application or not.

Declaration: `Property ConsoleApplication : Boolean`

Visibility: public

Access: Read

Description: `ConsoleApplication` returns `True` if the application is compiled as a console application (the default) or `False` if not. The result of this property is determined at compile-time by the settings of the compiler: it returns the value of the IsConsole (**??**) constant.

See also: IsConsole (**??**)

13.4.25 TCustomApplication.Location

Synopsis: Application location.

Declaration: `Property Location : string`

Visibility: public

Access: Read

Description: `Location` returns the directory part of the application binary. This property works on most platforms, although some platforms do not allow to retrieve this information (Mac OS for example has no reliable way to get this information). See the discussion of Paramstr (**??**) in the RTL documentation.

See also: Paramstr (**??**), Params (375)

13.4.26 TCustomApplication.Params

Synopsis: Command-line parameters.

Declaration: `Property Params[Index: Integer]: string`

Visibility: public

Access: Read

Description: `Params` gives access to the command-line parameters. They contain the value of the `Index`-th parameter, where `Index` runs from 0 to ParamCount (376). It is equivalent to calling ParamStr (**??**).

See also: ParamCount (376), Paramstr (**??**)

13.4.27 TCustomApplication.ParamCount

Synopsis: Number of command-line parameters.

Declaration: `Property ParamCount : Integer`

Visibility: public

Access: Read

Description: `ParamCount` returns the number of command-line parameters that were passed to the program. The actual parameters can be retrieved with the Params (375) property.

See also: Params (375), Paramstr (**??**), ParamCount (**??**)

13.4.28 TCustomApplication.EnvironmentVariable

Synopsis: Environment variable access.

Declaration: `Property EnvironmentVariable[envName: string]: string`

Visibility: public

Access: Read

Description: `EnvironmentVariable` gives access to the environment variables of the application: It returns the value of the environment variable `EnvName`, or an empty string if no such value is available.

To use this property, the name of the environment variable must be known. To get a list of available names (and values), GetEnvironmentList (373) can be used.

See also: GetEnvironmentList (373), TCustomApplication.Params (375)

13.4.29 TCustomApplication.OptionChar

Synopsis: Command-line switch character.

Declaration: `Property OptionChar : Char`

Visibility: public

Access: Read,Write

Description: `OptionChar` is the character used for command line switches. By default, this is the dash ('-') character, but it can be set to any other non-alphanumerical character (although no check is performed on this).

See also: FindOptionIndex (370), GetOptionValue (370), HasOption (371), CaseSensitiveOptions (376), CheckOptions (371)

13.4.30 TCustomApplication.CaseSensitiveOptions

Synopsis: Are options interpreted case sensitive or not.

Declaration: `Property CaseSensitiveOptions : Boolean`

Visibility: public

Access: Read,Write

Description: `CaseSensitiveOptions` determines whether FindOptionIndex (370) and CheckOptions (371) perform searches in a case sensitive manner or not. By default, the search is case-sensitive. Setting this property to `False` makes the search case-insensitive.

See also: FindOptionIndex (370), GetOptionValue (370), HasOption (371), OptionChar (376), CheckOptions (371)

13.4.31 TCustomApplication.CaseSensitiveShortOptions

Declaration: `Property CaseSensitiveShortOptions : Boolean`

Visibility: public

Access: Read,Write

13.4.32 TCustomApplication.CaseSensitiveLongOptions

Declaration: `Property CaseSensitiveLongOptions : Boolean`

Visibility: public

Access: Read,Write

13.4.33 TCustomApplication.StopOnException

Synopsis: Should the program loop stop on an exception.

Declaration: `Property StopOnException : Boolean`

Visibility: public

Access: Read,Write

Description: `StopOnException` controls the behaviour of the Run (369) and HandleException (368) procedures in case of an unhandled exception in the `DoRun` code. If `StopOnException` is `True` then Terminate (369) will be called after the exception was handled.

See also: Run (369), HandleException (368), Terminate (369)

13.4.34 TCustomApplication.ExceptionExitCode

Synopsis: ExitCode to use then terminating the program due to an exception.

Declaration: `Property ExceptionExitCode : LongInt`

Visibility: public

Access: Read,Write

Description: `ExceptionExitCode` is the exit code that will be passed to TCustomApplication.Terminate (369)

13.4.35 TCustomApplication.EventLogFilter

Synopsis: Event to filter events, before they are sent to the system log.

Declaration: `Property EventLogFilter : TEventLogTypes`

Visibility: public

Access: Read,Write

Description: `EventLogFilter` can be set to a set of event types that should be logged to the system log. If the set is empty, all event types are sent to the system log. If the set is non-empty, the TCustomApplication.Log (373) routine will check if the log event type is in the set, and if not, will not send the message to the system log.

See also: TCustomApplication.Log (373)

13.4.36 TCustomApplication.SingleInstance

Synopsis: Single instance used to control single application instance behaviour.

Declaration: `Property SingleInstance : TBaseSingleInstance`

Visibility: public

Access: Read

Description: `SingleInstance` is used when TCustomApplication.SingleInstanceEnabled (378) is set to `True`. It can be used to send a message to an already running instance, or to check for messages if the current instance is the sole ("server") instance running.

See also: TCustomApplication.SingleInstanceClass (378), TCustomApplication.SingleInstanceEnabled (378)

13.4.37 TCustomApplication.SingleInstanceClass

Synopsis: Class to use when creating single instance.

Declaration: `Property SingleInstanceClass : TBaseSingleInstanceClass`

Visibility: public

Access: Read,Write

Description: `SingleInstanceClass` can be used to set the class used to instantiate SingleInstance (378). The default class is determined by the global singleinstance default class as specified in #fcl.singleinstance.DefaultSingleInstanceCla (1482).

See also: TCustomApplication.SingleInstance (378), DefaultSingleInstanceClass (1482)

13.4.38 TCustomApplication.SingleInstanceEnabled

Synopsis: Enable single application instance control.

Declaration: `Property SingleInstanceEnabled : Boolean`

Visibility: public

Access: Read,Write

Description: `SingleInstanceEnabled` can be set to `true` to start single-instance application control. This will instantiate TCustomApplication.SingleInstance (378) using TCustomApplication.SingleInstanceClass (378) and starts the check to wee whether this application is a client or server instance.

See also: TCustomApplication.SingleInstance (378), TCustomApplication.SingleInstanceClass (378)

Chapter 14

Reference for unit 'custweb'

14.1 Used units

Table 14.1: Used units by unit 'custweb'

Name	Page
Classes	??
CustApp	365
eventlog	658
fphttp	778
HTTPDefs	1141
System	??
sysutils	??

14.2 Overview

The `custweb` unit provides the base functionality common to all HTTP hosting implementations in Free Pascal: CGI, FastCGI, HTTP server and Apache loadable modules. It contains the foundational classes TWebHandler (388) and TCustomWebApplication (382) that handle HTTP requests and provide web application functionality.

The unit supports module-based request handling, error management, event logging, and various configuration options for different hosting environments. It provides the infrastructure for routing requests to appropriate HTTP modules and managing the complete request-response cycle.

14.3 Constants, types and variables

14.3.1 Types

```
TGetModuleEvent = procedure(Sender: TObject; ARequest: TRequest;
  var ModuleClass: TCustomHTTPModuleClass
  )
                              of object
```

Event handler type called when the web application needs to determine which HTTP module class to use for handling a specific request. This allows custom logic for module selection based on request properties.

```
TInitModuleEvent = procedure(Sender: TObject; Module: TCustomHTTPModule
  )
                                of object
```

Event handler type called when an HTTP module is initialized before handling a request. Allows custom initialization of module properties and setup.

```
TLogEvent = procedure(EventType: TEventType; const Msg: string)
  of object
```

Event handler type called when the web application needs to log an event. Allows custom logging implementation for web application events and errors.

```
TOnShowRequestException = procedure(AResponse: TResponse;
  AnException: Exception;
  var handled: Boolean)
```

Event handler type called when an exception occurs during HTTP request processing. Allows custom handling of exceptions before the default error response is generated.

14.4 Procedures and functions

14.4.1 ExceptionToHTML

Synopsis: Utility function to format exceptions as HTML error pages

Declaration:
```
procedure ExceptionToHTML(S: TStrings; const E: Exception;
                          const Title: string; const Email: string;
                          const Administrator: string)
```

Visibility: default

Description: Formats an exception into a complete HTML error page with stack trace information. Generates a user-friendly error page that includes exception details, stack trace, and administrator contact information.

The generated HTML includes the application title, error message, stack trace with function names, and optional administrator email contact link.

Errors: No specific errors during HTML generation.

See also: TWebHandler.ShowRequestException (388)

14.5 EFPWebError

14.5.1 Description

`EFpWebError` is an exception class specifically for web application errors. It extends `EFPHTTPError` to provide HTTP-specific error handling with status codes and status text.

See also: TWebHandler.DoError (388)

14.6 TCustomWebApplication

14.6.1 Description

`TCustomWebApplication` extends `TCustomApplication` to provide web-specific functionality. It integrates a TWebHandler (388) instance for processing HTTP requests and provides properties for configuring web application behavior.

This class serves as the foundation for all web application implementations including CGI, FastCGI, and other HTTP hosting environments. It handles application lifecycle, event logging, and web handler management.

See also: TWebHandler (388)

14.6.2 Method overview

Page	Method	Description
382	Create	Creates and initializes a new web application instance
383	CreateForm	Creates a component instance with the web handler as owner
383	Destroy	Destroys the web application and cleans up resources
383	Initialize	Initializes the web application with error handling enabled
384	Terminate	Terminates the web application and web handler

14.6.3 Property overview

Page	Properties	Access	Description
386	Administrator	rw	Administrator name for error pages
385	AllowDefaultModule	rw	Controls whether requests without module names use default modules
385	ApplicationURL	rw	Base URL for the web application
385	DefaultModuleName	rw	Name of the default HTTP module
386	Email	rw	Administrator email address for error notifications
387	EventLog	r	Event log instance for logging web application events
384	HandleGetOnPost	rw	Controls whether GET requests are handled by POST handlers
387	LegacyRouting	rw	Controls whether to use legacy request routing
385	ModuleVariable	rw	Query parameter name for specifying HTTP module names
386	OnGetModule	rw	Event for custom module selection logic
386	OnShowRequestException	rw	Event for custom exception handling during request processing
387	OnUnknownRequestEncoding	rw	Event for handling unknown request content encodings
387	PreferModuleName	rw	Controls module name extraction preference
384	RedirectOnError	rw	Controls whether errors trigger HTTP redirects
384	RedirectOnErrorURL	rw	URL template for error redirects

14.6.4 TCustomWebApplication.Create

Synopsis: Creates and initializes a new web application instance

Declaration: `constructor Create(AOwner: TComponent);  Override`

Visibility: public

Description: Creates a new `TCustomWebApplication` instance and initializes the web handler through the abstract `InitializeWebHandler` method. Sets up event handlers for termination and logging.

Errors: May raise exceptions during web handler initialization.

See also: InitializeWebHandler (382), Destroy (383)

14.6.5 TCustomWebApplication.Destroy

Synopsis: Destroys the web application and cleans up resources

Declaration: `destructor Destroy;  Override`

Visibility: public

Description: Destroys the web application instance, freeing the web handler and event log resources. Performs proper cleanup of all web application components.

Errors: No specific errors expected during destruction.

See also: Create (382)

14.6.6 TCustomWebApplication.CreateForm

Synopsis: Creates a component instance with the web handler as owner

Declaration: `procedure CreateForm(AClass: TComponentClass; out Reference)`

Visibility: public

Description: Creates an instance of the specified component class with the web handler as the owner component. Provides a way to create forms and components within the web application context.

Errors: May raise exceptions during component creation.

14.6.7 TCustomWebApplication.Initialize

Synopsis: Initializes the web application with error handling enabled

Declaration: `procedure Initialize;  Override`

Visibility: public

Description: Initializes the web application and enables `StopOnException` to ensure proper error handling. Calls the inherited initialization from `TCustomApplication`.

Errors: May raise exceptions during application initialization.

14.6.8 TCustomWebApplication.Terminate

Synopsis: Terminates the web application and web handler

Declaration: `procedure Terminate;  Override`

Visibility: public

Description: Terminates the web application by first calling the inherited termination and then terminating the web handler if it's still running. Ensures clean shutdown of the web request processing loop.

Errors: No specific errors expected during termination.

See also: TWebHandler.Terminate (388)

14.6.9 TCustomWebApplication.HandleGetOnPost

Synopsis: Controls whether GET requests are handled by POST handlers

Declaration: `Property HandleGetOnPost : Boolean`

Visibility: public

Access: Read,Write

Description: Delegates to the TWebHandler.HandleGetOnPost (391) property. When `True`, allows POST request handlers to also process GET requests.

See also: TWebHandler.HandleGetOnPost (391)

14.6.10 TCustomWebApplication.RedirectOnError

Synopsis: Controls whether errors trigger HTTP redirects

Declaration: `Property RedirectOnError : Boolean`

Visibility: public

Access: Read,Write

Description: Delegates to the TWebHandler.RedirectOnError (391) property. When `True`, exceptions during request processing cause an HTTP redirect.

See also: TWebHandler.RedirectOnError (391)

14.6.11 TCustomWebApplication.RedirectOnErrorURL

Synopsis: URL template for error redirects

Declaration: `Property RedirectOnErrorURL : string`

Visibility: public

Access: Read,Write

Description: Delegates to the TWebHandler.RedirectOnErrorURL (391) property. Specifies the URL pattern used for error redirects.

See also: TWebHandler.RedirectOnErrorURL (391)

14.6.12 TCustomWebApplication.ApplicationURL

Synopsis: Base URL for the web application

Declaration: `Property ApplicationURL : string`

Visibility: public

Access: Read,Write

Description: Delegates to the TWebHandler.ApplicationURL (392) property. Specifies the base URL for the web application.

See also: TWebHandler.ApplicationURL (392)

14.6.13 TCustomWebApplication.AllowDefaultModule

Synopsis: Controls whether requests without module names use default modules

Declaration: `Property AllowDefaultModule : Boolean`

Visibility: public

Access: Read,Write

Description: Delegates to the TWebHandler.AllowDefaultModule (392) property. When `True`, requests without module names use the default module.

See also: TWebHandler.AllowDefaultModule (392)

14.6.14 TCustomWebApplication.DefaultModuleName

Synopsis: Name of the default HTTP module

Declaration: `Property DefaultModuleName : string`

Visibility: public

Access: Read,Write

Description: Delegates to the TWebHandler.DefaultModuleName (392) property. Specifies the default HTTP module for requests without explicit module names.

See also: TWebHandler.DefaultModuleName (392)

14.6.15 TCustomWebApplication.ModuleVariable

Synopsis: Query parameter name for specifying HTTP module names

Declaration: `Property ModuleVariable : string`

Visibility: public

Access: Read,Write

Description: Delegates to the TWebHandler.ModuleVariable (392) property. Specifies the query parameter name used to determine module names.

See also: TWebHandler.ModuleVariable (392)

14.6.16 TCustomWebApplication.OnGetModule

Synopsis: Event for custom module selection logic

Declaration: `Property OnGetModule : TGetModuleEvent`

Visibility: public

Access: Read,Write

Description: Delegates to the TWebHandler.OnGetModule (393) property. Event called to determine which HTTP module class should handle a request.

See also: TWebHandler.OnGetModule (393)

14.6.17 TCustomWebApplication.Email

Synopsis: Administrator email address for error notifications

Declaration: `Property Email : string`

Visibility: public

Access: Read,Write

Description: Delegates to the TWebHandler.Email (393) property. Email address displayed in error pages for user contact.

See also: TWebHandler.Email (393)

14.6.18 TCustomWebApplication.Administrator

Synopsis: Administrator name for error pages

Declaration: `Property Administrator : string`

Visibility: public

Access: Read,Write

Description: Delegates to the TWebHandler.Administrator (393) property. Name of the application administrator displayed in error pages.

See also: TWebHandler.Administrator (393)

14.6.19 TCustomWebApplication.OnShowRequestException

Synopsis: Event for custom exception handling during request processing

Declaration: `Property OnShowRequestException : TOnShowRequestException`

Visibility: public

Access: Read,Write

Description: Delegates to the TWebHandler.OnShowRequestException (394) property. Event called when an exception occurs during HTTP request processing.

See also: TWebHandler.OnShowRequestException (394)

14.6.20 TCustomWebApplication.OnUnknownRequestEncoding

Synopsis: Event for handling unknown request content encodings

Declaration: `Property OnUnknownRequestEncoding : TOnUnknownEncodingEvent`

Visibility: public

Access: Read,Write

Description: Delegates to the TWebHandler.OnUnknownRequestEncoding (394) property. Event called when request contains content with unknown encoding.

See also: TWebHandler.OnUnknownRequestEncoding (394)

14.6.21 TCustomWebApplication.EventLog

Synopsis: Event log instance for logging web application events

Declaration: `Property EventLog : TEventLog`

Visibility: public

Access: Read

Description: Provides access to the event log used by the web application for logging events and errors. The event log is created automatically when first accessed and configured for system logging.

See also: CreateEventLog (382)

14.6.22 TCustomWebApplication.PreferModuleName

Synopsis: Controls module name extraction preference

Declaration: `Property PreferModuleName : Boolean`

Visibility: public

Access: Read,Write

Description: Delegates to the TWebHandler.PreferModuleName (395) property. Controls whether module name extraction is preferred over path info parsing.

See also: TWebHandler.PreferModuleName (395)

14.6.23 TCustomWebApplication.LegacyRouting

Synopsis: Controls whether to use legacy request routing

Declaration: `Property LegacyRouting : Boolean`

Visibility: public

Access: Read,Write

Description: Delegates to the TWebHandler.LegacyRouting (395) property. When `True`, uses legacy routing instead of the HTTP router system.

See also: TWebHandler.LegacyRouting (395)

14.7 TWebHandler

14.7.1 Description

`TWebHandler` is the core component that handles HTTP requests in web applications. It provides the framework for routing requests to appropriate HTTP modules, handling errors, and managing the request-response cycle. This class serves as the foundation for all HTTP hosting implementations including CGI, FastCGI, HTTP server and Apache modules.

The handler supports both legacy routing and new HTTP routing systems, module lifecycle management, error handling with custom exception display, and event logging.

See also: TCustomWebApplication (382)

14.7.2 Method overview

Page	Method	Description
389	Create	Creates a new web handler instance
389	Destroy	Destroys the web handler instance
390	DoHandleRequest	Handles a complete HTTP request with error handling and cleanup
391	HandleRequest	Routes and processes an HTTP request to the appropriate handler
390	Log	Logs an event message using the configured log handler
390	Run	Main request processing loop for the web handler

14.7.3 Property overview

Page	Properties	Access	Description
393	Administrator	rw	Administrator name for error pages
392	AllowDefaultModule	rw	Controls whether requests without module names use default modules
392	ApplicationURL	rw	Base URL for the web application
392	DefaultModuleName	rw	Name of the default HTTP module for requests without explicit module names
393	Email	rw	Administrator email address for error notifications
391	HandleGetOnPost	rw	Controls whether GET requests are handled by POST handlers
395	LegacyRouting	rw	Controls whether to use legacy request routing instead of HTTP router
392	ModuleVariable	rw	Query parameter name for specifying HTTP module names
393	OnGetModule	rw	Event for custom module selection logic
394	OnIdle	rw	Event triggered when the handler is idle between requests
395	OnInitModule	rw	Event for initializing HTTP modules before request processing
394	OnLog	rw	Event for logging web application events and errors
394	OnShowRequestException	rw	Event for custom exception handling during request processing
394	OnUnknownRequestEncoding	rw	Event for handling unknown request content encodings
395	PreferModuleName	rw	Controls whether module name extraction is preferred over path info parsing
391	RedirectOnError	rw	Controls whether errors trigger HTTP redirects
391	RedirectOnErrorURL	rw	URL template for error redirects
393	Title	rw	Application title displayed in error pages

14.7.4 TWebHandler.Create

Synopsis: Creates a new web handler instance

Declaration: `constructor Create(AOwner: TComponent);  Override`

Visibility: public

Description: Creates and initializes a new `TWebHandler` instance with default settings. Sets up default module variable name, enables default module handling, and configures request handling options.

Errors: May raise exceptions during component initialization.

See also: Destroy (389)

14.7.5 TWebHandler.Destroy

Synopsis: Destroys the web handler instance

Declaration: `destructor Destroy;  Override`

Visibility: public

Description: Cleans up resources and destroys the `TWebHandler` instance. Resets module factory event handlers and performs standard component cleanup.

Errors: No specific errors expected during destruction.

See also: Create (389)

14.7.6 TWebHandler.Run

Synopsis: Main request processing loop for the web handler

Declaration: `procedure Run;  Virtual`

Visibility: public

Description: Starts the main request processing loop that waits for incoming HTTP requests and handles them. Continues processing until the handler is terminated, calling idle events between requests when available.

Errors: May raise exceptions during request waiting or processing.

See also: DoHandleRequest (390), OnIdle (394)

14.7.7 TWebHandler.Log

Synopsis: Logs an event message using the configured log handler

Declaration: `procedure Log(EventType: TEventType; const Msg: string)`

Visibility: public

Description: Logs an event message with the specified event type if a log event handler is assigned. Provides centralized logging for web application events and errors.

Errors: No specific errors, depends on log handler implementation.

See also: OnLog (394)

14.7.8 TWebHandler.DoHandleRequest

Synopsis: Handles a complete HTTP request with error handling and cleanup

Declaration: `procedure DoHandleRequest(ARequest: TRequest; AResponse: TResponse)`

Visibility: public

Description: Processes a complete HTTP request including error handling, response sending, and proper cleanup of request and response objects. Ensures content is sent even if not explicitly sent by the handler, and logs any content sending errors.

Errors: Handles and logs content sending errors internally.

See also: HandleRequest (391), EndRequest (388)

14.7.9 TWebHandler.HandleRequest

Synopsis: Routes and processes an HTTP request to the appropriate handler

Declaration:
```
procedure HandleRequest(ARequest: TRequest; AResponse: TResponse)
                        ; Virtual
```

Visibility: public

Description: Routes an HTTP request to the appropriate handler using either legacy routing or the HTTP router system. Catches and handles any exceptions that occur during request processing by calling the exception handler.

Errors: Catches all exceptions during request processing and handles them via `ShowRequestException`.

See also: LegacyRouting (395), ShowRequestException (388)

14.7.10 TWebHandler.HandleGetOnPost

Synopsis: Controls whether GET requests are handled by POST handlers

Declaration:
```
Property HandleGetOnPost : Boolean
```

Visibility: public

Access: Read,Write

Description: When `True`, allows POST request handlers to also process GET requests. This provides flexibility for web applications that need to handle both request types with the same logic.

14.7.11 TWebHandler.RedirectOnError

Synopsis: Controls whether errors trigger HTTP redirects

Declaration:
```
Property RedirectOnError : Boolean
```

Visibility: public

Access: Read,Write

Description: When `True`, exceptions during request processing cause an HTTP redirect to the URL specified in RedirectOnErrorURL (391). When `False`, errors are displayed as HTML error pages.

See also: RedirectOnErrorURL (391)

14.7.12 TWebHandler.RedirectOnErrorURL

Synopsis: URL template for error redirects

Declaration:
```
Property RedirectOnErrorURL : string
```

Visibility: public

Access: Read,Write

Description: Specifies the URL pattern used for error redirects when RedirectOnError (391) is `True`. The URL can contain format specifiers to include error message details in the redirect.

See also: RedirectOnError (391)

14.7.13 TWebHandler.ApplicationURL

Synopsis: Base URL for the web application

Declaration: `Property ApplicationURL : string`

Visibility: public

Access: Read,Write

Description: Specifies the base URL for the web application. When empty, the application URL is determined from the request script name. Used for constructing absolute URLs and setting module base URLs.

See also: SetBaseURL (388)

14.7.14 TWebHandler.AllowDefaultModule

Synopsis: Controls whether requests without module names use default modules

Declaration: `Property AllowDefaultModule : Boolean`

Visibility: public

Access: Read,Write

Description: When `True`, requests that don't specify a module name will use the default module specified in DefaultModuleName (392). When `False`, requests without module names result in an error.

See also: DefaultModuleName (392)

14.7.15 TWebHandler.DefaultModuleName

Synopsis: Name of the default HTTP module for requests without explicit module names

Declaration: `Property DefaultModuleName : string`

Visibility: public

Access: Read,Write

Description: Specifies which HTTP module to use when a request doesn't explicitly specify a module name and AllowDefaultModule (392) is `True`. If empty and only one module is registered, that module becomes the default.

See also: AllowDefaultModule (392)

14.7.16 TWebHandler.ModuleVariable

Synopsis: Query parameter name for specifying HTTP module names

Declaration: `Property ModuleVariable : string`

Visibility: public

Access: Read,Write

Description: Specifies the name of the query parameter used to determine which HTTP module should handle a request. Defaults to 'Module'. For example, '?Module=MyModule' would route to the MyModule HTTP module.

See also: GetModuleName (388)

14.7.17 TWebHandler.OnGetModule

Synopsis: Event for custom module selection logic

Declaration: `Property OnGetModule : TGetModuleEvent`

Visibility: public

Access: Read,Write

Description: Event called to determine which HTTP module class should handle a request. Allows custom module selection logic based on request properties, overriding the default module name resolution.

See also: TGetModuleEvent (380)

14.7.18 TWebHandler.Email

Synopsis: Administrator email address for error notifications

Declaration: `Property Email : string`

Visibility: public

Access: Read,Write

Description: Email address displayed in error pages for user contact. Used in HTML error pages to provide a contact point for error reporting.

See also: Administrator (393)

14.7.19 TWebHandler.Title

Synopsis: Application title displayed in error pages

Declaration: `Property Title : string`

Visibility: public

Access: Read,Write

Description: Title of the web application used in error page headers and titles. Provides branding and identification in error responses.

See also: ExceptionToHTML (381)

14.7.20 TWebHandler.Administrator

Synopsis: Administrator name for error pages

Declaration: `Property Administrator : string`

Visibility: public

Access: Read,Write

Description: Name of the application administrator displayed in error pages. Used together with the email address to provide contact information for error reporting.

See also: Email (393)

14.7.21 TWebHandler.OnShowRequestException

Synopsis: Event for custom exception handling during request processing

Declaration: `Property OnShowRequestException : TOnShowRequestException`

Visibility: public

Access: Read,Write

Description: Event called when an exception occurs during HTTP request processing. Allows custom handling of exceptions before the default error response is generated.

See also: TOnShowRequestException (381)

14.7.22 TWebHandler.OnIdle

Synopsis: Event triggered when the handler is idle between requests

Declaration: `Property OnIdle : TNotifyEvent`

Visibility: public

Access: Read,Write

Description: Event called between request processing cycles when the handler is waiting for new requests. Allows background processing or cleanup tasks during idle periods.

See also: Run (390)

14.7.23 TWebHandler.OnLog

Synopsis: Event for logging web application events and errors

Declaration: `Property OnLog : TLogEvent`

Visibility: public

Access: Read,Write

Description: Event called when the web application needs to log events or errors. Allows custom logging implementation for web application diagnostics.

See also: TLogEvent (381), Log (390)

14.7.24 TWebHandler.OnUnknownRequestEncoding

Synopsis: Event for handling unknown request content encodings

Declaration: `Property OnUnknownRequestEncoding : TOnUnknownEncodingEvent`

Visibility: public

Access: Read,Write

Description: Event called when the HTTP request contains content with an unknown or unsupported encoding. Allows custom handling of request content encoding issues.

14.7.25 TWebHandler.OnInitModule

Synopsis: Event for initializing HTTP modules before request processing

Declaration: `Property OnInitModule : TInitModuleEvent`

Visibility: public

Access: Read,Write

Description: Event called when an HTTP module is initialized before processing a request. Allows custom setup and configuration of module properties.

See also: TInitModuleEvent (381)

14.7.26 TWebHandler.PreferModuleName

Synopsis: Controls whether module name extraction is preferred over path info parsing

Declaration: `Property PreferModuleName : Boolean`

Visibility: public

Access: Read,Write

Description: When `True`, the handler prefers extracting module names from path info even when path contains multiple segments. Affects how module names are determined from request URLs.

See also: GetModuleName (388)

14.7.27 TWebHandler.LegacyRouting

Synopsis: Controls whether to use legacy request routing instead of HTTP router

Declaration: `Property LegacyRouting : Boolean`

Visibility: public

Access: Read,Write

Description: When `True`, uses the original request routing mechanism instead of the HTTP router system. Provides backward compatibility with older web applications.

See also: HandleRequest (391)

Chapter 15

Reference for unit 'daemonapp'

15.1 Used units

Table 15.1: Used units by unit 'daemonapp'

Name	Page
Classes	??
CustApp	365
eventlog	658
rtlconsts	??
System	??
sysutils	??

15.2 Overview

The daemonapp unit implements a `TApplication` class which encapsulates a daemon or service application. It handles installation where this is necessary, and does instantiation of the various daemons where necessary.

The unit consists of 3 separate classes which cooperate tightly:

TDaemon This is a class that implements the daemon's functionality. One or more descendents of this class can be implemented and instantiated in a single daemon application. For more information, see TDaemon (414).

TDaemonApplication This is the actual daemon application class. A global instance of this class is instantiated. It handles the command-line arguments, and instantiates the various daemons. For more information, see TDaemonApplication (418).

TDaemonDef This class defines the daemon in the operation system. The `TDaemonApplication` class has a collection of `TDaemonDef` instances, which it uses to start the various daemons. For more information, see TDaemonDef (421).

As can be seen, a single application can implement one ore more daemons (services). Each daemon will be run in a separate thread which is controlled by the application class.

The classes take care of logging through the TEventLog (660) class.

Many options are needed only to make the application behave as a windows service application on windows. These options are ignored in UNIX-like environment. The documentation will mention this.

15.3 Constants, types and variables

15.3.1 Resource strings

```
SControlFailed = 'Control code %s handling failed: %s'
```

The control code was not handled correctly.

```
SCustomCode = '[Custom code %d]'
```

A custom code was received.

```
SDaemonStatus = 'Daemon %s current status: %s'
```

Daemon status report log message.

```
SErrApplicationAlreadyCreated =
  'An application instance of class %s was already created.'
```

A second application instance is created.

```
SErrDaemonStartFailed = 'Failed to start daemon %s : %s'
```

The application failed to start the daemon.

```
SErrDuplicateName = 'Duplicate daemon name: %s'
```

Duplicate service name.

```
SErrNoDaemonDefForStatus =
  '%s: No daemon definition for status report'
```

Internal error: no daemon definition to report status for.

```
SErrNoDaemonForStatus = '%s: No daemon for status report'
```

Internal error: no daemon to report status for.

```
SErrNoServiceMapper = 'No daemon mapper class registered.'
```

No service mapper was found.

```
SErrNothingToDo = 'No command given, use ''%s -h'' for usage.'
```

No operation can be performed.

```
SErrOnlyOneMapperAllowed =
  'Not changing daemon mapper class %s with %s: Only 1 mapper allowed.'
```

An attempt was made to install a second service mapper.

```
SErrServiceManagerStartFailed =
  'Failed to start service manager: %s'
```

Unable to start or contact the service manager.

```
SErrUnknownDaemonClass = 'Unknown daemon class name: %s'
```

Unknown daemon class requested.

```
SErrWindowClass = 'Could not register window class'
```

Could not register window class.

```
SHelpCommand = 'Where command is one of the following:'
```

Options message displayed when writing help to the console.

```
SHelpInstall = 'To install the program as a service'
```

Install option message displayed when writing help to the console.

```
SHelpRun = 'To run the service'
```

Run option message displayed when writing help to the console.

```
SHelpUnInstall = 'To uninstall the service'
```

Uninstall option message displayed when writing help to the console.

```
SHelpUsage = 'Usage: %s [command]'
```

Usage message displayed when writing help to the console.

15.3.2 Types

```
TCurrentStatus = (csStopped,csStartPending,csStopPending,csRunning
  ,
                  csContinuePending,csPausePending,csPaused)
```

Table 15.2: Enumeration values for type TCurrentStatus

Value	Explanation
csContinuePending	The daemon is continuing, but not yet running.
csPaused	The daemon is paused: running but not active.
csPausePending	The daemon is about to be paused.
csRunning	The daemon is running (it is operational).
csStartPending	The daemon is starting, but not yet fully running.
csStopped	The daemon is stopped, i.e. inactive.
csStopPending	The daemon is stopping, but not yet fully stopped.

`TCurrentStatus` indicates the current state of the daemon. It changes from one state to the next during the time the instance is active. The daemon application changes the state of the daemon, depending on signals it gets from the operating system, by calling the appropriate methods.

```
TCustomControlCodeEvent = procedure(Sender: TCustomDaemon;
  ACode: DWord; var Handled: Boolean
  )
                                      of object
```

In case the system sends a non-standard control code to the daemon, an event handler is executed with this prototype.

```
TCustomControlCodeEvEvent = procedure(Sender: TCustomDaemon;
  ACode: DWord; AEventType: DWord
  ;
                                       AEventData: Pointer;
  var Handled: Boolean) of
  object
```

TCustomControlCodeEvEvent is the type used for the OnControlCodeEvent property in TDaemon.

```
TCustomDaemonApplicationClass = Class of TCustomDaemonApplication
```

Class pointer for `TCustomDaemonApplication`.

```
TCustomDaemonClass = Class of TCustomDaemon
```

The class type is needed in the TDaemonDef (421) definition.

```
TCustomDaemonMapperClass = Class of TCustomDaemonMapper
```

`TCustomDaemonMapperClass` is the class of `TCustomDaemonMapper`. It is used in the RegisterDaemonMapper (403) call.

```
TDaemonClass = Class of TDaemon
```

Class type of `TDaemon`.

```
TDaemonEvent = procedure(Sender: TCustomDaemon) of object
```

`TDaemonEvent` is used in event handling. The `Sender` is the TCustomDaemon (404) instance that has initiated the event.

```
TDaemonOKEvent = procedure(Sender: TCustomDaemon; var OK: Boolean
  )
                              of object
```

`TDaemonOKEvent` is used in event handling, when a boolean result must be obtained, for instance, to see if an operation was performed successfully.

```
TDaemonOption = (doAllowStop,doAllowPause,doInteractive)
```

Table 15.3: Enumeration values for type TDaemonOption

Value	Explanation
doAllowPause	The daemon can be paused.
doAllowStop	The daemon can be stopped.
doInteractive	The daemon interacts with the desktop.

Enumerated that enumerates the various daemon operation options.

```
TDaemonOptions = Set of TDaemonOption
```

`TDaemonOption` enumerates the various options a daemon can have.

```
TDaemonRunMode = (drmUnknown,drmInstall,drmUninstall,drmRun)
```

Table 15.4: Enumeration values for type TDaemonRunMode

Value	Explanation
drmInstall	Daemon install mode (windows only).
drmRun	Daemon is running normally.
drmUninstall	Daemon uninstall mode (windows only).
drmUnknown	Unknown mode.

`TDaemonRunMode` indicates in what mode the daemon application (as a whole) is currently running.

```
TErrorSeverity = (esIgnore,esNormal,esSevere,esCritical)
```

Table 15.5: Enumeration values for type TErrorSeverity

Value	Explanation
esCritical	Error is logged, and startup is stopped if last known good configuration is active, or system is restarted using last
esIgnore	Ignore startup errors.
esNormal	Error is logged, but startup continues.
esSevere	Error is logged, and startup is continued if last known good configuration is active, or system is restarted using la

`TErrorSeverity` determines what action windows takes when the daemon fails to start. It is used on windows only, and is ignored on other platforms.

```
TGuiLoopEvent = procedure of object
```

`TGuiLoopEvent` is the main GUI loop event procedure prototype. It is called by the application instance in case the daemon has a visual part, which needs to handle visual events. It is run in the main application thread.

```
TServiceType = (stWin32,stDevice,stFileSystem)
```

Table 15.6: Enumeration values for type TServiceType

Value	Explanation
stDevice	Device driver.
stFileSystem	File system driver.
stWin32	Regular win32 service.

The type of service. This type is used on windows only, to signal the operating system what kind of service is being installed or run.

```
TStartType = (stBoot,stSystem,stAuto,stManual,stDisabled)
```

Table 15.7: Enumeration values for type TStartType

Value	Explanation
stAuto	Started automatically by service manager during system startup.
stBoot	During system boot.
stDisabled	Service is not started, it is disabled.
stManual	Started manually by the user or other processes.
stSystem	During load of device drivers.

`TStartType` can be used to define when the service must be started on windows. This type is not used on other platforms.

```
TWinControlCode = (wccNetBindChange,wccParamChange,wccPreShutdown
  ,
                   wccShutdown,wccHardwareProfileChange,wccPowerEvent
  ,
                   wccSessionChange,wccTimeChange,wccTriggerEvent
  ,
                   wccUserModeReboot)
```

Table 15.8: Enumeration values for type TWinControlCode

Value	Explanation
wccHardwareProfileChange	
wccNetBindChange	
wccParamChange	
wccPowerEvent	
wccPreShutdown	
wccSessionChange	
wccShutdown	
wccTimeChange	
wccTriggerEvent	
wccUserModeReboot	

```
TWinControlCodes = Set of TWinControlCode
```

15.3.3 Variables

```
AppClass : TCustomDaemonApplicationClass
```

`AppClass` can be set to the class of a TCustomDaemonApplication (406) descendant. When the Application (402) function needs to create an application instance, this class will be used. If `Application` was already called, the value of `AppClass` will be ignored.

```
CurrentStatusNames : Array[TCurrentStatus] of string = ('Stopped'
  , 'Start Pending', 'Stop Pending', 'Running', 'Continue Pending',
  'Pause Pending', 'Paused')
```

Names for various service statuses.

```
DefaultDaemonOptions : TDaemonOptions = [doAllowStop, doAllowPause
  ]
```

`DefaultDaemonOptions` are the default options with which a daemon definition (TDaemonDef (421)) is created.

```
SStatus : Array[1..5] of string = ('Stop', 'Pause', 'Continue', 'Interrogate'
  , 'Shutdown')
```

Status message.

15.4 Procedures and functions

15.4.1 Application

Synopsis: Application instance.

Declaration: `function Application : TCustomDaemonApplication`

Visibility: default

Description: `Application` is the TCustomDaemonApplication (406) instance used by this application. The instance is created at the first invocation of this function, so it is possible to use RegisterDaemonApplicationClass (403) to register an alternative `TCustomDaemonApplication` class to run the application.

See also: TCustomDaemonApplication (406), RegisterDaemonApplicationClass (403)

15.4.2 DaemonError

Synopsis: Raise an `EDaemon` exception.

Declaration:
```
procedure DaemonError(const Msg: string)
procedure DaemonError(const Fmt: string; const Args: Array of const)
```

Visibility: default

Description: `DaemonError` raises an EDaemon (403) exception with message `Msg` or it formats the message using `Fmt` and `Args`.

See also: EDaemon (403)

15.4.3 RegisterDaemonApplicationClass

Synopsis: Register alternative `TCustomDaemonApplication` class.

Declaration:
```
procedure RegisterDaemonApplicationClass
                                        (AClass: TCustomDaemonApplicationClass)
```

Visibility: default

Description: `RegisterDaemonApplicationClass` can be used to register an alternative TCustomDaemonApplication (406) descendent which will be used when creating the global Application (402) instance. Only the last registered class pointer will be used.

See also: TCustomDaemonApplication (406), Application (402)

15.4.4 RegisterDaemonClass

Synopsis: Register daemon.

Declaration:
```
procedure RegisterDaemonClass(AClass: TCustomDaemonClass)
```

Visibility: default

Description: `RegisterDaemonClass` must be called for each TCustomDaemon (404) descendent that is used in the class: the class pointer and class name are used by the TCustomDaemonMapperClass (399) class to create a `TCustomDaemon` instance when a daemon is required.

See also: TCustomDaemonMapperClass (399), TCustomDaemon (404)

15.4.5 RegisterDaemonMapper

Synopsis: Register a daemon mapper class.

Declaration:
```
procedure RegisterDaemonMapper(AMapperClass: TCustomDaemonMapperClass)
```

Visibility: default

Description: `RegisterDaemonMapper` can be used to register an alternative class for the global daemonmapper. The daemonmapper will be used only when the application is being run, by the TCustomDaemonApplication (406) code, so registering an alternative mapping class should happen in the initialization section of the application units.

See also: TCustomDaemonApplication (406), TCustomDaemonMapperClass (399)

15.5 EDaemon

15.5.1 Description

`EDaemon` is the exception class used by all code in the DaemonApp unit.

See also: DaemonError (402)

15.6 TCustomDaemon

15.6.1 Description

`TCustomDaemon` implements all the basic calls that are needed for a daemon to function. Descendents of `TCustomDaemon` can override these calls to implement the daemon-specific behaviour.

`TCustomDaemon` is an abstract class, it should never be instantiated. Either a descendent of it must be created and instantiated, or a descendent of TDaemon (414) can be designed to implement the behaviour of the daemon.

See also: TDaemon (414), TDaemonDef (421), TDaemonController (418), TDaemonApplication (418)

15.6.2 Method overview

Page	Method	Description
404	CheckControlMessages	
404	LogMessage	Log a message to the system log.
405	ReportStatus	Report the current status to the operating system.

15.6.3 Property overview

Page	Properties	Access	Description
406	Controller	r	`TDaemonController` instance controlling this daemon instance.
405	DaemonThread	r	Thread in which daemon is running.
405	Definition	r	The definition used to instantiate this daemon instance.
406	Logger	r	`TEventLog` instance used to send messages to the system log.
406	Status	rw	Current status of the daemon.

15.6.4 TCustomDaemon.CheckControlMessages

Synopsis:

Declaration: `procedure CheckControlMessages(Wait: Boolean)`

Visibility: public

Description:

15.6.5 TCustomDaemon.LogMessage

Synopsis: Log a message to the system log.

Declaration: `procedure LogMessage(const Msg: string)`

Visibility: public

Description: `LogMessage` can be used to send a message `Msg` to the system log. A TEventLog (660) instance is used to actually send messages to the system log.

The message is sent with an 'error' flag (using TEventLog.Error (664)).

Errors: None.

See also: ReportStatus (405)

15.6.6 TCustomDaemon.ReportStatus

Synopsis: Report the current status to the operating system.

Declaration: `procedure ReportStatus`

Visibility: public

Description: `ReportStatus` can be used to report the current status to the operating system. The start and stop or pause and continue operations can be slow to start up. This call can (and should) be used to report the current status to the operating system during such lengthy operations, or else it may conclude that the daemon has died.

This call is mostly important on windows operating systems, to notify the service manager that the operation is still in progress.

The implementation of `ReportStatus` simply calls `ReportStatus` in the controller.

Errors: None.

See also: LogMessage (404)

15.6.7 TCustomDaemon.Definition

Synopsis: The definition used to instantiate this daemon instance.

Declaration: `Property Definition : TDaemonDef`

Visibility: public

Access: Read

Description: `Definition` is the TDaemonDef (421) definition that was used to start the daemon instance. It can be used to retrieve additional information about the intended behaviour of the daemon.

See also: TDaemonDef (421)

15.6.8 TCustomDaemon.DaemonThread

Synopsis: Thread in which daemon is running.

Declaration: `Property DaemonThread : TThread`

Visibility: public

Access: Read

Description: `DaemonThread` is the thread in which the daemon instance is running. Each daemon instance in the application runs in it's own thread, none of which are the main thread of the application. The application main thread is used to handle control messages coming from the operating system.

See also: Controller (406)

15.6.9 TCustomDaemon.Controller

Synopsis: `TDaemonController` instance controlling this daemon instance.

Declaration: `Property Controller : TDaemonController`

Visibility: public

Access: Read

Description: `Controller` points to the `TDaemonController` instance that was created by the application instance to control this daemon.

See also: DaemonThread (405)

15.6.10 TCustomDaemon.Status

Synopsis: Current status of the daemon.

Declaration: `Property Status : TCurrentStatus`

Visibility: public

Access: Read,Write

Description: `Status` indicates the current status of the daemon. It is set by the various operations that the controller operates on the daemon, and should not be set manually.

`Status` is the value which `ReportStatus` will send to the operating system.

See also: ReportStatus (405)

15.6.11 TCustomDaemon.Logger

Synopsis: `TEventLog` instance used to send messages to the system log.

Declaration: `Property Logger : TEventLog`

Visibility: public

Access: Read

Description: `Logger` is the TEventLog (660) instance used to send messages to the system log. It is used by the LogMessage (404) call, but is accessible through the `Logger` property in case more configurable logging is needed than offered by `LogMessage`.

See also: LogMessage (404), TEventLog (660)

15.7 TCustomDaemonApplication

15.7.1 Description

`TCustomDaemonApplication` is a TCustomApplication (366) descendent which is the main application instance for a daemon. It handles the command-line and decides what to do when the application is started, depending on the command-line options given to the application, by calling the various methods.

It creates the necessary TDaemon (414) instances by checking the TCustomDaemonMapperClass (399) instance that contains the daemon maps.

See also: TCustomApplication (366), TCustomDaemonMapperClass (399)

15.7.2 Method overview

Page	Method	Description
407	Create	Constructor for the class instance.
408	CreateDaemon	Create daemon instance.
409	CreateForm	Create a component.
407	Destroy	Clean up the `TCustomDaemonApplication` instance.
408	InstallDaemons	Install all daemons.
408	RunDaemons	Run all daemons.
407	ShowException	Show an exception.
409	ShowHelp	Display a help message.
408	StopDaemons	Stop all daemons.
409	UnInstallDaemons	Uninstall all daemons.

15.7.3 Property overview

Page	Properties	Access	Description
411	AutoRegisterMessageFile	rw	Automatically register the message file.
410	EventLog	r	Event logger instance.
410	GuiHandle	rw	Handle of GUI loop main application window handle.
410	GUIMainLoop	rw	GUI main loop callback.
409	OnRun	rw	Event executed when the daemon is run.
410	RunMode	r	Application mode.

15.7.4 TCustomDaemonApplication.Create

Synopsis: Constructor for the class instance.

Declaration: `constructor Create(AOwner: TComponent);  Override`

Visibility: public

Description: Constructor for the class instance.

15.7.5 TCustomDaemonApplication.Destroy

Synopsis: Clean up the `TCustomDaemonApplication` instance.

Declaration: `destructor Destroy;  Override`

Visibility: public

Description: `Destroy` cleans up the event log instance and then calls the inherited destroy.

See also: TCustomDaemonApplication.EventLog (410)

15.7.6 TCustomDaemonApplication.ShowException

Synopsis: Show an exception.

Declaration: `procedure ShowException(E: Exception);  Override`

Visibility: public

Description: `ShowException` is overridden by `TCustomDaemonApplication`, it sends the exception message to the system log.

15.7.7 TCustomDaemonApplication.CreateDaemon

Synopsis: Create daemon instance.

Declaration: `function CreateDaemon(DaemonDef: TDaemonDef) : TCustomDaemon`

Visibility: public

Description: `CreateDaemon` is called whenever a TCustomDaemon (404) instance must be created from a TDaemonDef (421) daemon definition, passed in `DaemonDef`. It initializes the `TCustomDaemon` instance, and creates a controller instance of type TDaemonController (418) to control the daemon. Finally, it assigns the created daemon to the TDaemonDef.Instance (423) property.

Errors: In case of an error, an exception may be raised.

See also: TDaemonController (418), TCustomDaemon (404), TDaemonDef (421), TDaemonDef.Instance (423)

15.7.8 TCustomDaemonApplication.StopDaemons

Synopsis: Stop all daemons.

Declaration: `procedure StopDaemons(Force: Boolean)`

Visibility: public

Description: `StopDaemons` sends the `STOP` control code to all daemons, or the `SHUTDOWN` control code in case `Force` is `True`.

See also: TDaemonController.Controller (420), TCustomDaemonApplication.UnInstallDaemons (409), TCustomDaemonApplication.RunDaemons (408)

15.7.9 TCustomDaemonApplication.InstallDaemons

Synopsis: Install all daemons.

Declaration: `procedure InstallDaemons`

Visibility: public

Description: `InstallDaemons` installs all known daemons, i.e. registers them with the service manager on Windows. This method is called if the application is run with the `-i` or `-install` or `/install` command-line option.

See also: TCustomDaemonApplication.UnInstallDaemons (409), TCustomDaemonApplication.RunDaemons (408), TCustomDaemonApplication.StopDaemons (408)

15.7.10 TCustomDaemonApplication.RunDaemons

Synopsis: Run all daemons.

Declaration: `procedure RunDaemons`

Visibility: public

Description: `RunDaemons` runs (starts) all known daemons. This method is called if the application is run with the `-r` or `-run` methods.

See also: TCustomDaemonApplication.UnInstallDaemons (409), TCustomDaemonApplication.InstallDaemons (408), TCustomDaemonApplication.StopDaemons (408)

15.7.11 TCustomDaemonApplication.UnInstallDaemons

Synopsis: Uninstall all daemons.

Declaration: `procedure UnInstallDaemons`

Visibility: public

Description: `UnInstallDaemons` uninstalls all known daemons, i.e. deregisters them with the service manager on Windows. This method is called if the application is run with the `-u` or `-uninstall` or `/uninstall` command-line option.

See also: TCustomDaemonApplication.RunDaemons (408), TCustomDaemonApplication.InstallDaemons (408), TCustomDaemonApplication.StopDaemons (408)

15.7.12 TCustomDaemonApplication.ShowHelp

Synopsis: Display a help message.

Declaration: `procedure ShowHelp;  Virtual`

Visibility: public

Description: `ShowHelp` displays a help message explaining the command-line options on standard output.

15.7.13 TCustomDaemonApplication.CreateForm

Synopsis: Create a component.

Declaration:
```
procedure CreateForm(InstanceClass: TComponentClass; var Reference)
                        ;  Virtual
```

Visibility: public

Description: `CreateForm` creates an instance of `InstanceClass` and fills `Reference` with the class instance pointer. It's main purpose is to give an IDE a means of assuring that forms or datamodules are created on application startup: the IDE will generate calls for all modules that are auto-created.

Errors: An exception may arise if the instance wants to stream itself from resources, but no resources are found.

See also: TCustomDaemonApplication.CreateDaemon (408)

15.7.14 TCustomDaemonApplication.OnRun

Synopsis: Event executed when the daemon is run.

Declaration: `Property OnRun : TNotifyEvent`

Visibility: public

Access: Read,Write

Description: `OnRun` is triggered when the daemon application is run and no appropriate options (one of install, uninstall or run) was given.

See also: TCustomDaemonApplication.RunDaemons (408), TCustomDaemonApplication.InstallDaemons (408), TCustomDaemonApplication.UnInstallDaemons (409)

15.7.15 TCustomDaemonApplication.EventLog

Synopsis: Event logger instance.

Declaration: `Property EventLog : TEventLog`

Visibility: public

Access: Read

Description: `EventLog` is the TEventLog (660) instance which is used to log events to the system log. It is created when the application instance is created, and destroyed when the application is destroyed.

See also: TEventLog (660)

15.7.16 TCustomDaemonApplication.GUIMainLoop

Synopsis: GUI main loop callback.

Declaration: `Property GUIMainLoop : TGuiLoopEvent`

Visibility: public

Access: Read,Write

Description: `GUIMainLoop` contains a reference to a method that can be called to process a main GUI loop. The procedure should return only when the main GUI has finished and the application should exit. It is called when the daemons are running.

See also: TCustomDaemonApplication.GuiHandle (410)

15.7.17 TCustomDaemonApplication.GuiHandle

Synopsis: Handle of GUI loop main application window handle.

Declaration: `Property GuiHandle : THandle`

Visibility: public

Access: Read,Write

Description: `GuiHandle` is the handle of a GUI window which can be used to run a message handling loop on. It is created when no GUIMainLoop (410) procedure exists, and the application creates and runs a message loop by itself.

See also: GUIMainLoop (410)

15.7.18 TCustomDaemonApplication.RunMode

Synopsis: Application mode.

Declaration: `Property RunMode : TDaemonRunMode`

Visibility: public

Access: Read

Description: `RunMode` indicates in which mode the application is running currently. It is set automatically by examining the command-line, and when set, one of InstallDaemons (408), RunDaemons (408) or UnInstallDaemons (409) is called.

See also: InstallDaemons (408), RunDaemons (408), UnInstallDaemons (409)

15.7.19 TCustomDaemonApplication.AutoRegisterMessageFile

Synopsis: Automatically register the message file.

Declaration: `Property AutoRegisterMessageFile : Boolean`

Visibility: public

Access: Read,Write

Description: `AutoRegisterMessageFile` can be set to `True` to automatically register the service binary as the source of resource strings for the event viewer.

The event log mechanism uses several resource strings in the fclel.res file. These resource strings must be registered in the windows event viewer. Setting this property to `True` takes case of this registration when the program is started.

15.8 TCustomDaemonMapper

15.8.1 Description

The `TCustomDaemonMapper` class is responsible for mapping a daemon definition to an actual `TDaemon` instance. It maintains a TDaemonDefs (425) collection with daemon definitions, which can be used to map the definition of a daemon to a `TDaemon` descendent class.

An IDE such as Lazarus can design a `TCustomDaemonMapper` instance visually, to help establish the relationship between various TDaemonDef (421) definitions and the actual TDaemon (414) instances that will be used to run the daemons.

The `TCustomDaemonMapper` class has no support for streaming. The TDaemonMapper (427) class has support for streaming (and hence visual designing).

See also: TDaemon (414), TDaemonDef (421), TDaemonDefs (425), TDaemonMapper (427)

15.8.2 Method overview

Page	Method	Description
411	Create	Create a new instance of `TCustomDaemonMapper`.
412	Destroy	Clean up and destroy a `TCustomDaemonMapper` instance.

15.8.3 Property overview

Page	Properties	Access	Description
412	DaemonDefs	rw	Collection of daemons.
412	OnCreate	rw	Event called when the daemon mapper is created.
413	OnDestroy	rw	Event called when the daemon mapper is freed.
413	OnInstall	rw	Event called when the daemons are installed.
413	OnRun	rw	Event called when the daemons are executed.
413	OnUnInstall	rw	Event called when the daemons are uninstalled.

15.8.4 TCustomDaemonMapper.Create

Synopsis: Create a new instance of `TCustomDaemonMapper`.

Declaration: `constructor Create(AOwner: TComponent); Override`

Visibility: public

Description: `Create` creates a new instance of a `TCustomDaemonMapper`. It creates the TDaemonDefs (425) collection and then calls the inherited constructor. It should never be necessary to create a daemon mapper manually, the application will create a global `TCustomDaemonMapper` instance.

See also: TDaemonDefs (425), TCustomDaemonApplication (406), TCustomDaemonMapper.Destroy (412)

15.8.5 TCustomDaemonMapper.Destroy

Synopsis: Clean up and destroy a `TCustomDaemonMapper` instance.

Declaration: `destructor Destroy;  Override`

Visibility: public

Description: `Destroy` frees the DaemonDefs (412) collection and calls the inherited destructor.

See also: TDaemonDefs (425), TCustomDaemonMapper.Create (411)

15.8.6 TCustomDaemonMapper.DaemonDefs

Synopsis: Collection of daemons.

Declaration: `Property DaemonDefs : TDaemonDefs`

Visibility: published

Access: Read,Write

Description: `DaemonDefs` is the application's global collection of daemon definitions. This collection will be used to decide at runtime which `TDaemon` class must be created to run or install a daemon.

See also: TCustomDaemonApplication (406)

15.8.7 TCustomDaemonMapper.OnCreate

Synopsis: Event called when the daemon mapper is created.

Declaration: `Property OnCreate : TNotifyEvent`

Visibility: published

Access: Read,Write

Description: `OnCreate` is an event that is called when the `TCustomDaemonMapper` instance is created. It can for instance be used to dynamically create daemon definitions at runtime.

See also: OnDestroy (413), OnUnInstall (413), OnCreate (412), OnDestroy (413)

15.8.8 TCustomDaemonMapper.OnDestroy

Synopsis: Event called when the daemon mapper is freed.

Declaration: `Property OnDestroy : TNotifyEvent`

Visibility: published

Access: Read,Write

Description: `OnDestroy` is called when the global daemon mapper instance is destroyed. it can be used to release up any resources that were allocated when the instance was created, in the OnCreate (412) event.

See also: OnCreate (412), OnInstall (413), OnUnInstall (413), OnCreate (412)

15.8.9 TCustomDaemonMapper.OnRun

Synopsis: Event called when the daemons are executed.

Declaration: `Property OnRun : TNotifyEvent`

Visibility: published

Access: Read,Write

Description: `OnRun` is the event called when the daemon application is executed to run the daemons (with command-line parameter '-r'). it is called exactly once.

See also: OnInstall (413), OnUnInstall (413), OnCreate (412), OnDestroy (413)

15.8.10 TCustomDaemonMapper.OnInstall

Synopsis: Event called when the daemons are installed.

Declaration: `Property OnInstall : TNotifyEvent`

Visibility: published

Access: Read,Write

Description: `OnInstall` is the event called when the daemon application is executed to install the daemons (with command-line parameter '-i' or '/install'). it is called exactly once.

See also: OnRun (413), OnUnInstall (413), OnCreate (412), OnDestroy (413)

15.8.11 TCustomDaemonMapper.OnUnInstall

Synopsis: Event called when the daemons are uninstalled.

Declaration: `Property OnUnInstall : TNotifyEvent`

Visibility: published

Access: Read,Write

Description: `OnUnInstall` is the event called when the daemon application is executed to uninstall the daemons (with command-line parameter '-u' or '/uninstall'). it is called exactly once.

See also: OnRun (413), OnInstall (413), OnCreate (412), OnDestroy (413)

15.9 TDaemon

15.9.1 Description

`TDaemon` is a TCustomDaemon (404) descendent which is meant for development in a visual environment: it contains event handlers for all major operations. Whenever a `TCustomDaemon` method is executed, it's execution is shunted to the event handler, which can be filled with code in the IDE.

All the events of the daemon are executed in the thread in which the daemon's controller is running (as given by DaemonThread (405)), which is not the main program thread.

See also: TCustomDaemon (404), TDaemonController (418)

15.9.2 Property overview

Page	Properties	Access	Description
417	AfterInstall	rw	Called after the daemon was installed.
417	AfterUnInstall	rw	Called after the daemon is uninstalled.
416	BeforeInstall	rw	Called before the daemon will be installed.
417	BeforeUnInstall	rw	Called before the daemon is uninstalled.
414	Definition		
415	OnContinue	rw	Daemon continue.
417	OnControlCode	rw	Called when a control code is received for the daemon.
418	OnControlCodeEvent	rw	
416	OnExecute	rw	Daemon execute event.
415	OnPause	rw	Daemon pause event.
416	OnShutDown	rw	Daemon shutdown.
414	OnStart	rw	Daemon start event.
415	OnStop	rw	Daemon stop event.
414	Status		

15.9.3 TDaemon.Definition

Declaration: `Property Definition :`

Visibility: public

Access:

15.9.4 TDaemon.Status

Declaration: `Property Status :`

Visibility: public

Access:

15.9.5 TDaemon.OnStart

Synopsis: Daemon start event.

Declaration: `Property OnStart : TDaemonOKEvent`

Visibility: published

Access: Read,Write

Description: `OnStart` is the event called when the daemon must be started. This event handler should return as quickly as possible. If it must perform lengthy operations, it is best to report the status to the operating system at regular intervals using the ReportStatus (405) method.

If the start of the daemon should do some continuous action, then this action should be performed in a new thread: this thread should then be created and started in the OnExecute (416) event handler, so the event handler can return at once.

See also: TDaemon.OnStop (415), TDaemon.OnExecute (416), TDaemon.OnContinue (415), ReportStatus (405)

15.9.6 TDaemon.OnStop

Synopsis: Daemon stop event.

Declaration: `Property OnStop : TDaemonOKEvent`

Visibility: published

Access: Read,Write

Description: `OnStart` is the event called when the daemon must be stopped. This event handler should return as quickly as possible. If it must perform lengthy operations, it is best to report the status to the operating system at regular intervals using the ReportStatus (405) method.

If a thread was started in the OnExecute (416) event, this is the place where the thread should be stopped.

See also: TDaemon.OnStart (414), TDaemon.OnPause (415), ReportStatus (405)

15.9.7 TDaemon.OnPause

Synopsis: Daemon pause event.

Declaration: `Property OnPause : TDaemonOKEvent`

Visibility: published

Access: Read,Write

Description: `OnPause` is the event called when the daemon must be stopped. This event handler should return as quickly as possible. If it must perform lengthy operations, it is best to report the status to the operating system at regular intervals using the ReportStatus (405) method.

If a thread was started in the OnExecute (416) event, this is the place where the thread's execution should be suspended.

See also: TDaemon.OnStop (415), TDaemon.OnContinue (415), ReportStatus (405)

15.9.8 TDaemon.OnContinue

Synopsis: Daemon continue.

Declaration: `Property OnContinue : TDaemonOKEvent`

Visibility: published

Access: Read,Write

Description: `OnPause` is the event called when the daemon must be stopped. This event handler should return as quickly as possible. If it must perform lengthy operations, it is best to report the status to the operating system at regular intervals using the ReportStatus (405) method.

If a thread was started in the OnExecute (416) event and it was suspended in a OnPause (414) event, this is the place where the thread's executed should be resumed.

See also: TDaemon.OnStart (414), TDaemon.OnPause (415), ReportStatus (405)

15.9.9 TDaemon.OnShutDown

Synopsis: Daemon shutdown.

Declaration: `Property OnShutDown : TDaemonEvent`

Visibility: published

Access: Read,Write

Description: `OnShutDown` is the event called when the daemon must be shut down. When the system is being shut down and the daemon does not respond to stop signals, then a shutdown message is sent to the daemon. This event can be used to respond to such a message. The daemon process will simply be stopped after this event.

If a thread was started in the OnExecute (416), this is the place where the thread's executed should be stopped or the thread freed from memory.

See also: TDaemon.OnStart (414), TDaemon.OnPause (415), ReportStatus (405)

15.9.10 TDaemon.OnExecute

Synopsis: Daemon execute event.

Declaration: `Property OnExecute : TDaemonEvent`

Visibility: published

Access: Read,Write

Description: `OnExecute` is executed once after the daemon was started. If assigned, it should perform whatever operation the daemon is designed.

If the daemon's action is event based, then no `OnExecute` handler is needed, and the events will control the daemon's execution: the daemon thread will then go in a loop, passing control messages to the daemon.

If an `OnExecute` event handler is present, the checking for control messages must be done by the implementation of the `OnExecute` handler.

See also: TDaemon.OnStart (414), TDaemon.OnStop (415)

15.9.11 TDaemon.BeforeInstall

Synopsis: Called before the daemon will be installed.

Declaration: `Property BeforeInstall : TDaemonEvent`

Visibility: published

Access: Read,Write

Description: `BeforeInstall` is called before the daemon is installed. It can be done to specify extra dependencies, or change the daemon description etc.

See also: AfterInstall (417), BeforeUnInstall (417), AfterUnInstall (417)

15.9.12 TDaemon.AfterInstall

Synopsis: Called after the daemon was installed.

Declaration: `Property AfterInstall : TDaemonEvent`

Visibility: published

Access: Read,Write

Description: `AfterInstall` is called after the daemon was successfully installed.

See also: BeforeInstall (416), BeforeUnInstall (417), AfterUnInstall (417)

15.9.13 TDaemon.BeforeUnInstall

Synopsis: Called before the daemon is uninstalled.

Declaration: `Property BeforeUnInstall : TDaemonEvent`

Visibility: published

Access: Read,Write

Description: `BeforeUnInstall` is called before the daemon is uninstalled.

See also: BeforeInstall (416), AfterInstall (417), AfterUnInstall (417)

15.9.14 TDaemon.AfterUnInstall

Synopsis: Called after the daemon is uninstalled.

Declaration: `Property AfterUnInstall : TDaemonEvent`

Visibility: published

Access: Read,Write

Description: `AfterUnInstall` is called after the daemon is successfully uninstalled.

See also: BeforeInstall (416), AfterInstall (417), BeforeUnInstall (417)

15.9.15 TDaemon.OnControlCode

Synopsis: Called when a control code is received for the daemon.

Declaration: `Property OnControlCode : TCustomControlCodeEvent`

Visibility: published

Access: Read,Write

Description: `OnControlCode` is called when the daemon receives a control code. If the daemon has not handled the control code, it should set the `Handled` parameter to `False`. By default it is set to `True`.

See also: Architecture (396)

15.9.16 TDaemon.OnControlCodeEvent

Synopsis:

Declaration: `Property OnControlCodeEvent : TCustomControlCodeEvEvent`

Visibility: published

Access: Read,Write

Description:

15.10 TDaemonApplication

15.10.1 Description

`TDaemonApplication` is the default TCustomDaemonApplication (406) descendent that is used to run the daemon application. It is possible to register an alternative `TCustomDaemonApplication` class (using RegisterDaemonApplicationClass (403)) to run the application in a different manner.

See also: TCustomDaemonApplication (406), RegisterDaemonApplicationClass (403)

15.11 TDaemonController

15.11.1 Description

`TDaemonController` is a class that is used by the TDaemonApplication (418) class to control the daemon during runtime. The `TDaemonApplication` class instantiates an instance of `TDaemonController` for each daemon in the application and communicates with the daemon through the `TDaemonController` instance. It should rarely be necessary to access or use this class.

See also: TCustomDaemon (404), TDaemonApplication (418)

15.11.2 Method overview

Page	Method	Description
420	Controller	Controller.
419	Create	Create a new instance of the `TDaemonController` class.
419	Destroy	Free a `TDaemonController` instance.
419	Main	Daemon main entry point.
420	ReportStatus	Report the status to the operating system.
419	StartService	Start the service.

15.11.3 Property overview

Page	Properties	Access	Description
421	CheckPoint	r	Send checkpoint signal to the operating system.
420	Daemon	r	Daemon instance this controller controls.
421	LastStatus	r	Last reported status.
420	Params	r	Parameters passed to the daemon.

15.11.4 TDaemonController.Create

Synopsis: Create a new instance of the `TDaemonController` class.

Declaration: `constructor Create(AOwner: TComponent); Override`

Visibility: public

Description: `Create` creates a new instance of the `TDaemonController` class. It should never be necessary to create a new instance manually, because the controllers are created by the global TDaemonApplication (418) instance, and `AOwner` will be set to the global TDaemonApplication (418) instance.

See also: TDaemonApplication (418), Destroy (419)

15.11.5 TDaemonController.Destroy

Synopsis: Free a `TDaemonController` instance.

Declaration: `destructor Destroy; Override`

Visibility: public

Description: `Destroy` deallocates some resources allocated when the instance was created.

See also: Create (419)

15.11.6 TDaemonController.StartService

Synopsis: Start the service.

Declaration: `procedure StartService; Virtual`

Visibility: public

Description: `StartService` starts the service controlled by this instance.

Errors: None.

See also: TDaemonController.Main (419)

15.11.7 TDaemonController.Main

Synopsis: Daemon main entry point.

Declaration: `procedure Main(Argc: DWord; Args: PPChar); Virtual`

Visibility: public

Description: `Main` is the service's main entry point, called when the system wants to start the service. The global application will call this function whenever required, with the appropriate arguments.

The standard implementation starts the daemon thread, and waits for it to stop. All other daemon action - such as responding to control code events - is handled by the thread.

Errors: If the daemon thread cannot be created, an exception is raised.

See also: TDaemonThread (428)

15.11.8 TDaemonController.Controller

Synopsis: Controller.

Declaration:
```
procedure Controller(ControlCode: DWord; EventType: DWord;
                     EventData: Pointer);  Virtual
```

Visibility: public

Description: `Controller` is responsible for sending the control code to the daemon thread so it can be processed.

This routine is currently only used on windows, as there is no service manager on Linux. Later on this may be changed to respond to signals on Linux as well.

See also: TDaemon.OnControlCode (417)

15.11.9 TDaemonController.ReportStatus

Synopsis: Report the status to the operating system.

Declaration:
```
function ReportStatus : Boolean;  Virtual
```

Visibility: public

Description: `ReportStatus` reports the status of the daemon to the operating system. On windows, this sends the current service status to the service manager. On other operating systems, this sends a message to the system log.

Errors: If an error occurs, an error message is sent to the system log.

See also: TCustomDaemon.ReportStatus (405), TDaemonController.LastStatus (421)

15.11.10 TDaemonController.Daemon

Synopsis: Daemon instance this controller controls.

Declaration:
```
Property Daemon : TCustomDaemon
```

Visibility: public

Access: Read

Description: `Daemon` is the daemon instance that is controller by this instance of the `TDaemonController` class.

15.11.11 TDaemonController.Params

Synopsis: Parameters passed to the daemon.

Declaration:
```
Property Params : TStrings
```

Visibility: public

Access: Read

Description: `Params` contains the parameters passed to the daemon application by the operating system, comparable to the application's command-line parameters. The property is set by the Main (419) method.

15.11.12 TDaemonController.LastStatus

Synopsis: Last reported status.

Declaration: `Property LastStatus : TCurrentStatus`

Visibility: public

Access: Read

Description: `LastStatus` is the last status reported to the operating system.

See also: ReportStatus (420)

15.11.13 TDaemonController.CheckPoint

Synopsis: Send checkpoint signal to the operating system.

Declaration: `Property CheckPoint : DWord`

Visibility: public

Access: Read

Description: `CheckPoint` can be used to send a checkpoint signal during lengthy operations, to signal that a lengthy operation is in progress. This should be used mainly on windows, to signal the service manager that the service is alive.

See also: ReportStatus (420)

15.12 TDaemonDef

15.12.1 Description

`TDaemonDef` contains the definition of a daemon in the application: The name of the daemon, which TCustomDaemon (404) descendent should be started to run the daemon, a description, and various other options should be set in this class. The global `TDaemonApplication` instance maintains a collection of `TDaemonDef` instances and will use these definitions to install or start the various daemons.

See also: TDaemonApplication (418), TDaemon (414)

15.12.2 Method overview

Page	Method	Description
422	Create	Create a new `TDaemonDef` instance.
422	Destroy	Free a `TDaemonDef` from memory.

15.12.3 Property overview

Page	Properties	Access	Description
422	DaemonClass	r	`TDaemon` class to use for this daemon.
423	DaemonClassName	rw	Name of the `TDaemon` class to use for this daemon.
423	Description	rw	Description of the daemon.
424	DisplayName	rw	Displayed name of the daemon (service).
424	Enabled	rw	Is the daemon enabled or not.
423	Instance	rw	Instance of the daemon class.
425	LogStatusReport	rw	Log the status report to the system log.
423	Name	rw	Name of the daemon (service).
425	OnCreateInstance	rw	Event called when a daemon in instantiated.
424	Options	rw	Service options.
424	RunArguments	rw	Additional command-line arguments when running daemon.
425	WinBindings	rw	Windows-specific bindings (windows only).

15.12.4 TDaemonDef.Create

Synopsis: Create a new `TDaemonDef` instance.

Declaration:
```
constructor Create(ACollection: TCollection);  Override
```

Visibility: public

Description: `Create` initializes a new `TDaemonDef` instance. It should not be necessary to instantiate a definition manually, it is handled by the collection.

See also: TDaemonDefs (425)

15.12.5 TDaemonDef.Destroy

Synopsis: Free a `TDaemonDef` from memory.

Declaration:
```
destructor Destroy;  Override
```

Visibility: public

Description: `Destroy` removes the `TDaemonDef` from memory.

15.12.6 TDaemonDef.DaemonClass

Synopsis: `TDaemon` class to use for this daemon.

Declaration:
```
Property DaemonClass : TCustomDaemonClass
```

Visibility: public

Access: Read

Description: `DaemonClass` is the `TDaemon` class that is used when this service is requested. It is looked up in the application's global daemon mapper by it's name in DaemonClassName (423).

See also: DaemonClassName (423), TDaemonMapper (427)

15.12.7 TDaemonDef.Instance

Synopsis: Instance of the daemon class.

Declaration: `Property Instance : TCustomDaemon`

Visibility: public

Access: Read,Write

Description: `Instance` points to the TDaemon (414) instance that is used when the service is in operation at runtime.

See also: TDaemonDef.DaemonClass (422)

15.12.8 TDaemonDef.DaemonClassName

Synopsis: Name of the `TDaemon` class to use for this daemon.

Declaration: `Property DaemonClassName : string`

Visibility: published

Access: Read,Write

Description: `DaemonClassName` is the name of the `TCustomDaemon` class that will be used whenever the service is needed. The name is used to look up the class pointer registered in the daemon mapper, when TCustomDaemonApplication (406) creates an instance of the daemon.

See also: TDaemonDef.Instance (423), TDaemonDef.DaemonClass (422), RegisterDaemonClass (403), TCustomDaemon (404), TCustomDaemonApplication (406)

15.12.9 TDaemonDef.Name

Synopsis: Name of the daemon (service).

Declaration: `Property Name : string`

Visibility: published

Access: Read,Write

Description: `Name` is the internal name of the daemon as it is known to the operating system.

See also: TDaemonDef.DisplayName (424)

15.12.10 TDaemonDef.Description

Synopsis: Description of the daemon.

Declaration: `Property Description : string`

Visibility: published

Access: Read,Write

Description: `Description` is the description shown in the Windows service manager when managing this service. It is supplied to the windows service manager when the daemon is installed.

15.12.11 TDaemonDef.DisplayName

Synopsis: Displayed name of the daemon (service).

Declaration: `Property DisplayName : string`

Visibility: published

Access: Read,Write

Description: `DisplayName` is the displayed name of the daemon as it is known to the operating system.

See also: TDaemonDef.Name (423)

15.12.12 TDaemonDef.RunArguments

Synopsis: Additional command-line arguments when running daemon.

Declaration: `Property RunArguments : string`

Visibility: published

Access: Read,Write

Description: `RunArguments` specifies any additional command-line arguments that should be specified when running the daemon: these arguments will be passed to the service manager when registering the service on windows.

15.12.13 TDaemonDef.Options

Synopsis: Service options.

Declaration: `Property Options : TDaemonOptions`

Visibility: published

Access: Read,Write

Description: `Options` tells the operating system which operations can be performed on the daemon while it is running.

This option is only used during the installation of the daemon.

15.12.14 TDaemonDef.Enabled

Synopsis: Is the daemon enabled or not.

Declaration: `Property Enabled : Boolean`

Visibility: published

Access: Read,Write

Description: `Enabled` specifies whether a daemon should be installed, run or uninstalled. Disabled daemons are not installed, run or uninstalled.

15.12.15 TDaemonDef.WinBindings

Synopsis: Windows-specific bindings (windows only).

Declaration: `Property WinBindings : TWinBindings`

Visibility: published

Access: Read,Write

Description: `WinBindings` is used to group together the windows-specific properties of the daemon. This property is totally ignored on other platforms.

See also: TWinBindings (433)

15.12.16 TDaemonDef.OnCreateInstance

Synopsis: Event called when a daemon in instantiated.

Declaration: `Property OnCreateInstance : TNotifyEvent`

Visibility: published

Access: Read,Write

Description: `OnCreateInstance` is called whenever an instance of the daemon is created. This can be used for instance when a single `TDaemon` class is used to run several services, to correctly initialize the TDaemon.

15.12.17 TDaemonDef.LogStatusReport

Synopsis: Log the status report to the system log.

Declaration: `Property LogStatusReport : Boolean`

Visibility: published

Access: Read,Write

Description: `LogStatusReport` can be set to `True` to send the status reports also to the system log. This can be used to track the progress of the daemon.

See also: TCustomDaemon.ReportStatus (405)

15.13 TDaemonDefs

15.13.1 Description

`TDaemonDefs` is the class of the global list of daemon definitions. It contains an item for each daemon in the application.

Normally it is not necessary to create an instance of `TDaemonDefs` manually. The global TCustomDaemonMapper (411) instance will create a collection and maintain it.

See also: TCustomDaemonMapper (411), TDaemonDef (421)

15.13.2 Method overview

Page	Method	Description
426	Create	Create a new instance of a `TDaemonDefs` collection.
427	DaemonDefByName	Find and return instance of daemon definition with given name.
426	FindDaemonDef	Find and return instance of daemon definition with given name.
426	IndexOfDaemonDef	Return index of daemon definition.

15.13.3 Property overview

Page	Properties	Access	Description
427	Daemons	rw	Indexed access to `TDaemonDef` instances.

15.13.4 TDaemonDefs.Create

Synopsis: Create a new instance of a `TDaemonDefs` collection.

Declaration: `constructor Create(AOwner: TPersistent; AClass: TCollectionItemClass)`

Visibility: public

Description: `Create` creates a new instance of the `TDaemonDefs` collection. It keeps the `AOwner` parameter for future reference and calls the inherited constructor.

Normally it is not necessary to create an instance of `TDaemonDefs` manually. The global TCustomDaemonMapper (411) instance will create a collection and maintain it.

See also: TDaemonDef (421)

15.13.5 TDaemonDefs.IndexOfDaemonDef

Synopsis: Return index of daemon definition.

Declaration: `function IndexOfDaemonDef(const DaemonName: string) : Integer`

Visibility: public

Description: `IndexOfDaemonDef` searches the collection for a TDaemonDef (421) instance with a name equal to `DaemonName`, and returns it's index. It returns -1 if no definition was found with this name. The search is case insensitive.

See also: TDaemonDefs.FindDaemonDef (426), TDaemonDefs.DaemonDefByName (427)

15.13.6 TDaemonDefs.FindDaemonDef

Synopsis: Find and return instance of daemon definition with given name.

Declaration: `function FindDaemonDef(const DaemonName: string) : TDaemonDef`

Visibility: public

Description: `FindDaemonDef` searches the list of daemon definitions and returns the TDaemonDef (421) instance whose name matches `DaemonName`. If no definition is found, `Nil` is returned.

See also: TDaemonDefs.IndexOfDaemonDef (426), TDaemonDefs.DaemonDefByName (427)

15.13.7 TDaemonDefs.DaemonDefByName

Synopsis: Find and return instance of daemon definition with given name.

Declaration: `function DaemonDefByName(const DaemonName: string) : TDaemonDef`

Visibility: public

Description: `FindDaemonDef` searches the list of daemon definitions and returns the TDaemonDef (421) instance whose name matches `DaemonName`. If no definition is found, an EDaemon (403) exception is raised.

The FindDaemonDef (426) call does not raise an error, but returns `Nil` instead.

Errors: If no definition is found, an EDaemon (403) exception is raised.

See also: TDaemonDefs.IndexOfDaemonDef (426), TDaemonDefs.FindDaemonDef (426)

15.13.8 TDaemonDefs.Daemons

Synopsis: Indexed access to `TDaemonDef` instances.

Declaration: `Property Daemons[Index: Integer]: TDaemonDef; default`

Visibility: public

Access: Read,Write

Description: `Daemons` is the default property of `TDaemonDefs`, it gives access to the `TDaemonDef` instances in the collection.

See also: TDaemonDef (421)

15.14 TDaemonMapper

15.14.1 Description

`TDaemonMapper` is a direct descendent of TCustomDaemonMapper (411), but introduces no new functionality. It's sole purpose is to make it possible for an IDE to stream the `TDaemonMapper` instance.

For this purpose, it overrides the `Create` constructor and tries to find a resource with the same name as the class name, and tries to stream the instance from this resource.

If the instance should not be streamed, the CreateNew (428) constructor can be used instead.

See also: CreateNew (428), Create (428)

15.14.2 Method overview

Page	Method	Description
428	Create	Create a new `TDaemonMapper` instance and initializes it from streamed resources.
428	CreateNew	Create a new `TDaemonMapper` instance without initialization.

15.14.3 TDaemonMapper.Create

Synopsis: Create a new `TDaemonMapper` instance and initializes it from streamed resources.

Declaration:
```
constructor Create(AOwner: TComponent);  Override
```

Visibility: default

Description: `Create` initializes a new instance of `TDaemonMapper` and attempts to read the component from resources compiled in the application.

If the instance should not be streamed, the CreateNew (428) constructor can be used instead.

Errors: If no streaming system is found, or no resource exists for the class, an exception is raised.

See also: CreateNew (428)

15.14.4 TDaemonMapper.CreateNew

Synopsis: Create a new `TDaemonMapper` instance without initialization.

Declaration:
```
constructor CreateNew(AOwner: TComponent; Dummy: Integer)
```

Visibility: default

Description: `CreateNew` initializes a new instance of `TDaemonMapper`. In difference with the `Create` constructor, it does not attempt to read the component from a stream.

See also: Create (428)

15.15 TDaemonThread

15.15.1 Description

`TDaemonThread` is the thread in which the daemons in the application are run. Each daemon is run in it's own thread.

It should not be necessary to create these threads manually, the TDaemonController (418) class will take care of this.

See also: TDaemonController (418), TDaemon (414)

15.15.2 Method overview

Page	Method	Description
429	CheckControlMessage	Check if a control message has arrived.
430	ContinueDaemon	Continue the daemon.
429	Create	Create a new thread.
429	Execute	Run the daemon.
430	InterrogateDaemon	Report the daemon status.
430	PauseDaemon	Pause the daemon.
430	ShutDownDaemon	Shut down daemon.
429	StopDaemon	Stops the daemon.

15.15.3 Property overview

Page	Properties	Access	Description
431	Daemon	r	Daemon instance.

15.15.4 TDaemonThread.Create

Synopsis: Create a new thread.

Declaration: `constructor Create(ADaemon: TCustomDaemon)`

Visibility: public

Description: `Create` creates a new thread instance. It initializes the `Daemon` property with the passed `ADaemon`. The thread is created suspended.

See also: TDaemonThread.Daemon (431)

15.15.5 TDaemonThread.Execute

Synopsis: Run the daemon.

Declaration: `procedure Execute;  Override`

Visibility: public

Description: `Execute` starts executing the daemon and waits till the daemon stops. It also listens for control codes for the daemon.

See also: TDaemon.Execute (414)

15.15.6 TDaemonThread.CheckControlMessage

Synopsis: Check if a control message has arrived.

Declaration: `procedure CheckControlMessage(WaitForMessage: Boolean)`

Visibility: public

Description: `CheckControlMessage` checks if a control message has arrived for the daemon and executes the appropriate daemon message. If the parameter `WaitForMessage` is `True`, then the routine waits for the message to arrive. If it is `False` and no message is present, it returns at once.

15.15.7 TDaemonThread.StopDaemon

Synopsis: Stops the daemon.

Declaration: `function StopDaemon : Boolean;  Virtual`

Visibility: public

Description: `StopDaemon` attempts to stop the `Daemon` by calling methods in the `TCustomDaemon` instance. StopDaemon also terminates the thread. The return value is True if the Daemon was successfully stopped in the method.

See also: TDaemonThread.Daemon (431), TDaemonThread.PauseDaemon (430), TDaemonThread.ShutDownDaemon (430), TCustomDaemon (404), TThread.Terminate (**??**)

15.15.8 TDaemonThread.PauseDaemon

Synopsis: Pause the daemon.

Declaration: `function PauseDaemon : Boolean;  Virtual`

Visibility: public

Description: `PauseDaemon` attempts to pause the `Daemon` by calling methods in the TCustomDaemon (404) instance, and calling `Suspend` to suspend the thread. It returns `True` if the attempt was successful.

See also: TDaemonThread.StopDaemon (429), TDaemonThread.ContinueDaemon (430), TDaemonThread.ShutDownDaemon (430), TCustomDaemon (404), TThread.Suspend (**??**)

15.15.9 TDaemonThread.ContinueDaemon

Synopsis: Continue the daemon.

Declaration: `function ContinueDaemon : Boolean;  Virtual`

Visibility: public

Description: `ContinueDaemon` attempts to restart the `Daemon` by calling methods in the TCustomDaemon (404) instance. It returns `True` if the attempt was successful.

See also: TDaemonThread.Daemon (431), TDaemonThread.StopDaemon (429), TDaemonThread.PauseDaemon (430), TDaemonThread.ShutDownDaemon (430), TCustomDaemon (404)

15.15.10 TDaemonThread.ShutDownDaemon

Synopsis: Shut down daemon.

Declaration: `function ShutDownDaemon : Boolean;  Virtual`

Visibility: public

Description: `ShutDownDaemon` shuts down the `Daemon` for the thread. This happens normally only when the system is shut down and the daemon didn't respond to the stop request. The return valuie is the result from the method in the TCustomDaemon (404) instance. The thread is terminated in this method.

See also: TDaemonThread.StopDaemon (429), TDaemonThread.PauseDaemon (430), TDaemonThread.ContinueDaemon (430), TCustomDaemon (404), TThread.Terminate (**??**)

15.15.11 TDaemonThread.InterrogateDaemon

Synopsis: Report the daemon status.

Declaration: `function InterrogateDaemon : Boolean;  Virtual`

Visibility: public

Description: `InterrogateDaemon` simply calls TCustomDaemon.ReportStatus (405) for the daemon that is running in this thread. It always returns `True`.

See also: TCustomDaemon.ReportStatus (405)

15.15.12 TDaemonThread.Daemon

Synopsis: Daemon instance.

Declaration: `Property Daemon : TCustomDaemon`

Visibility: public

Access: Read

Description: `Daemon` is the daemon instance which is running in this thread.

See also: TDaemon (414)

15.16 TDependencies

15.16.1 Description

`TDependencies` is just a descendent of `TCollection` which contains a series of dependencies on other services. It overrides the default property of `TCollection` to return TDependency (432) instances.

See also: TDependency (432)

15.16.2 Method overview

Page	Method	Description
431	Create	Create a new instance of a `TDependencies` collection.

15.16.3 Property overview

Page	Properties	Access	Description
431	Items	rw	Default property override.

15.16.4 TDependencies.Create

Synopsis: Create a new instance of a `TDependencies` collection.

Declaration: `constructor Create(AOwner: TPersistent)`

Visibility: public

Description: `Create` Create a new instance of a `TDependencies` collection.

15.16.5 TDependencies.Items

Synopsis: Default property override.

Declaration: `Property Items[Index: Integer]: TDependency; default`

Visibility: public

Access: Read,Write

Description: `Items` overrides the default property of `TCollection` so the items are of type TDependency (432).

See also: TDependency (432)

15.17 TDependency

15.17.1 Description

`TDependency` is a collection item used to specify dependencies on other daemons (services) in windows. It is used only on windows and when installing the daemon: changing the dependencies of a running daemon has no effect.

See also: TDependencies (431), TDaemonDef (421)

15.17.2 Method overview

Page	Method	Description
432	Assign	Assign `TDependency` instance to another.

15.17.3 Property overview

Page	Properties	Access	Description
432	IsGroup	rw	Name refers to a service group.
432	Name	rw	Name of the service.

15.17.4 TDependency.Assign

Synopsis: Assign `TDependency` instance to another.

Declaration:
```
procedure Assign(Source: TPersistent);  Override
```

Visibility: public

Description: `Assign` is overridden by `TDependency` to copy all properties from one instance to another.

15.17.5 TDependency.Name

Synopsis: Name of the service.

Declaration:
```
Property Name : string
```

Visibility: published

Access: Read,Write

Description: `Name` is the name of a service or service group that the current daemon depends on.

See also: TDependency.IsGroup (432)

15.17.6 TDependency.IsGroup

Synopsis: Name refers to a service group.

Declaration:
```
Property IsGroup : Boolean
```

Visibility: published

Access: Read,Write

Description: `IsGroup` can be set to `True` to indicate that `Name` refers to the name of a service group.

See also: TDependency.Name (432)

15.18 TWinBindings

15.18.1 Description

`TWinBindings` contains windows-specific properties for the daemon definition (in TDaemonDef.WinBindings (425)). If the daemon should not run on Windows, then the properties can be ignored.

See also: TDaemonDef (421), TDaemonDef.WinBindings (425)

15.18.2 Method overview

Page	Method	Description
434	Assign	Copies all properties.
433	Create	Create a new `TWinBindings` instance.
433	Destroy	Remove a `TWinBindings` instance from memory.

15.18.3 Property overview

Page	Properties	Access	Description
437	AcceptedCodes	rw	
434	Dependencies	rw	Service dependencies.
434	ErrCode	rw	Service specific error code.
436	ErrorSeverity	rw	Error severity in case of startup failure.
435	GroupName	rw	Service group name.
436	IDTag	rw	Location in the service group.
435	Password	rw	Password for service startup.
436	ServiceType	rw	Type of service.
435	StartType	rw	Service startup type.
435	UserName	rw	Username to run service as.
436	WaitHint	rw	Timeout wait hint.
434	Win32ErrCode	rw	General windows error code.

15.18.4 TWinBindings.Create

Synopsis: Create a new `TWinBindings` instance.

Declaration: `constructor Create`

Visibility: public

Description: `Create` initializes various properties such as the dependencies.

See also: TDaemonDef (421), TDaemonDef.WinBindings (425), TWinBindings.Dependencies (434)

15.18.5 TWinBindings.Destroy

Synopsis: Remove a `TWinBindings` instance from memory.

Declaration: `destructor Destroy;  Override`

Visibility: public

Description: `Destroy` cleans up the `TWinBindings` instance.

See also: TWinBindings.Dependencies (434), TWinBindings.Create (433)

15.18.6 TWinBindings.Assign

Synopsis: Copies all properties.

Declaration: `procedure Assign(Source: TPersistent); Override`

Visibility: public

Description: `Assign` is overridden by `TWinBindings` so all properties are copied from `Source` to the `TWinBindings` instance.

15.18.7 TWinBindings.ErrCode

Synopsis: Service specific error code.

Declaration: `Property ErrCode : DWord`

Visibility: public

Access: Read,Write

Description: `ErrCode` contains a service specific error code that is reported with TCustomDaemon.ReportStatus (405) to the windows service manager. If it is zero, then the contents of Win32ErrCode (434) are reported. If it is nonzero, then the windows-errorcode is set to `ERROR_SERVICE_SPECIFIC_ERROR`.

See also: TWinBindings.Win32ErrCode (434)

15.18.8 TWinBindings.Win32ErrCode

Synopsis: General windows error code.

Declaration: `Property Win32ErrCode : DWord`

Visibility: public

Access: Read,Write

Description: `Win32ErrCode` is a general windows service error code that can be reported with TCustomDaemon.ReportStatus (405) to the windows service manager. It is sent if ErrCode (434) is zero.

See also: ErrCode (434)

15.18.9 TWinBindings.Dependencies

Synopsis: Service dependencies.

Declaration: `Property Dependencies : TDependencies`

Visibility: published

Access: Read,Write

Description: `Dependencies` contains the list of other services (or service groups) that this service depends on. Windows will first attempt to start these services prior to starting this service. If they cannot be started, then the service will not be started either.

This property is only used during installation of the service.

15.18.10 TWinBindings.GroupName

Synopsis: Service group name.

Declaration: `Property GroupName : string`

Visibility: published

Access: Read,Write

Description: `GroupName` specifies the name of a service group that the service belongs to. If it is empty, then the service does not belong to any group.

This property is only used during installation of the service.

See also: TDependency.IsGroup (432)

15.18.11 TWinBindings.Password

Synopsis: Password for service startup.

Declaration: `Property Password : string`

Visibility: published

Access: Read,Write

Description: `Password` contains the service password: if the service is started with credentials other than one of the system users, then the password for the user must be entered here.

This property is only used during installation of the service.

See also: UserName (435)

15.18.12 TWinBindings.UserName

Synopsis: Username to run service as.

Declaration: `Property UserName : string`

Visibility: published

Access: Read,Write

Description: `Username` specifies the name of a user whose credentials should be used to run the service. If it is left empty, the service is run as the system user. The password can be set in the Password (435) property.

This property is only used during installation of the service.

See also: Password (435)

15.18.13 TWinBindings.StartType

Synopsis: Service startup type.

Declaration: `Property StartType : TStartType`

Visibility: published

Access: Read,Write

Description: `StartType` specifies when the service should be started during system startup.

This property is only used during installation of the service.

15.18.14 TWinBindings.WaitHint

Synopsis: Timeout wait hint.

Declaration: `Property WaitHint : Integer`

Visibility: published

Access: Read,Write

Description: `WaitHint` specifies the estimated time for a start/stop/pause or continue operation (in milliseconds). Reportstatus should be called prior to this time to report the next status.

See also: TCustomDaemon.ReportStatus (405)

15.18.15 TWinBindings.IDTag

Synopsis: Location in the service group.

Declaration: `Property IDTag : DWord`

Visibility: published

Access: Read,Write

Description: `IDTag` contains the location of the service in the service group after installation of the service. It should not be set, it is reported by the service manager.

This property is only used during installation of the service.

15.18.16 TWinBindings.ServiceType

Synopsis: Type of service.

Declaration: `Property ServiceType : TServiceType`

Visibility: published

Access: Read,Write

Description: `ServiceType` specifies what kind of service is being installed.

This property is only used during installation of the service.

15.18.17 TWinBindings.ErrorSeverity

Synopsis: Error severity in case of startup failure.

Declaration: `Property ErrorSeverity : TErrorSeverity`

Visibility: published

Access: Read,Write

Description: `ErrorSeverity` can be used at installation time to tell the windows service manager how to behave when the service fails to start during system startup.

This property is only used during installation of the service.

15.18.18 TWinBindings.AcceptedCodes

Synopsis:

Declaration: `Property AcceptedCodes : TWinControlCodes`

Visibility: published

Access: Read,Write

Description:

Chapter 16

Reference for unit 'DB'

16.1 Used units

Table 16.1: Used units by unit 'DB'

Name	Page
Classes	??
FmtBCD	??
MaskUtils	1363
System	??
sysutils	??
Variants	??

16.2 Overview

The db unit provides the basis for all database access mechanisms. It introduces abstract classes, on which all database access mechanisms are based: TDataset (496) representing a set of records from a database, TField (549) which represents the contents of a field in a record, TDatasource (536) which acts as an event distributor on behalf of a dataset and TParams (631) which can be used to parameterize queries. The databases connections themselves are abstracted in the TDatabase (487) class.

16.3 Constants, types and variables

16.3.1 Constants

```
DefaultFieldClasses : Array[TFieldType] of TFieldClass = (TField,
  TStringField, TSmallIntField, TLongintField, TWordField, TBooleanField
  , TFloatField, TCurrencyField, TBCDField, TDateField, TTimeField,
  TDateTimeField, TBytesField, TVarBytesField, TAutoIncField, TBlobField
  , TMemoField, TGraphicField, TBlobField, TBlobField, TBlobField, TBlobField
  , Nil, TStringField, TWideStringField, TLargeIntField, Nil, TArrayField
  , Nil, Nil, TBlobField, TMemoField, TVariantField, Nil, Nil, TGuidField
  , Nil, TFMTBCDField, TWideStringField, TWideMemoField, Nil, Nil, TLongWordField
```

```
  , TShortintField, TByteField, TExtendedField, TSingleField)
```

DefaultFieldClasses contains the TField (549) descendent class to use when a TDataset instance needs to create fields based on the TFieldDefs (581) field definitions when opening the dataset. The entries can be set to create customized TField descendents for certain field datatypes in all datasets.

```
dsEditModes = [dsEdit, dsInsert, dsSetKey]
```

dsEditModes contains the various values of TDataset.State (524) for which the dataset is in edit mode, i.e. states in which it is possible to set field values for that dataset.

```
dsMaxBufferCount = MAXINT div 8
```

Maximum data buffers count for dataset.

```
dsMaxStringSize = 8192
```

Maximum size of string fields.

```
dsWriteModes = [dsEdit, dsInsert, dsSetKey, dsCalcFields, dsFilter
  , dsNewValue, dsInternalCalc, dsRefreshFields]
```

dsWriteModes contains the various values of TDataset.State (524) for which data can be written to the dataset buffer.

```
FieldTypeNames : Array[TFieldType] of string = ('Unknown', 'String'
  , 'Smallint', 'Integer', 'Word', 'Boolean', 'Float', 'Currency', 'BCD'
  , 'Date', 'Time', 'DateTime', 'Bytes', 'VarBytes', 'AutoInc', 'Blob'
  , 'Memo', 'Graphic', 'FmtMemo', 'ParadoxOle', 'DBaseOle', 'TypedBinary'
  , 'Cursor', 'FixedChar', 'WideString', 'Largeint', 'ADT', 'Array'
  , 'Reference', 'DataSet', 'OraBlob', 'OraClob', 'Variant', 'Interface'
  , 'IDispatch', 'Guid', 'TimeStamp', 'FMTBcd', 'FixedWideChar', 'WideMemo'
  , 'OraTimeStamp', 'OraInterval', 'LongWord', 'Shortint', 'Byte', 'Extended'
  , 'Single')
```

FieldTypeNames contains the names (in English) for the various field data types.

```
FieldTypetoVariantMap : Array[TFieldType] of Integer = (varError,
  varOleStr, varSmallint, varInteger, varSmallint, varBoolean, varDouble
  , varCurrency, varCurrency, varDate, varDate, varDate, varOleStr,
  varOleStr, varInteger, varOleStr, varOleStr, varOleStr, varOleStr
  , varOleStr, varOleStr, varOleStr, varError, varOleStr, varOleStr
  , varInt64, varError, varError, varError, varError, varOleStr, varOleStr
  , varVariant, varUnknown, varDispatch, varOleStr, varOleStr, varDouble
  , varOleStr, varOleStr, varUnknown, varUnknown, varLongWord, varShortint
  , varByte, varDouble, varSingle)
```

FieldTypetoVariantMap contains for each field datatype the variant value type that corresponds to it. If a field type cannot be expressed by a variant type, then varError is stored in the variant value.

```
ftBlobTypes = [ftBlob, ftMemo, ftGraphic, ftFmtMemo, ftParadoxOle
  , ftDBaseOle, ftTypedBinary, ftOraBlob, ftOraClob, ftWideMemo]
```

ftBlobTypes is a constant containing all blob field data types. It is to be preferred over the TBlobType (441) range, which contains some non-blob types as well.

```
ObjectFieldTypes = [ftADT, ftArray, ftReference, ftDataSet]
```

```
SQLDelimiterCharacters = [';', ',', ' ', '(', ')', #13, #10, #9]
```

SQL statement delimiter token characters.

```
YesNoChars : Array[Boolean] of Char = ('N', 'Y')
```

Array of characters mapping a boolean to Y/N.

16.3.2 Types

```
LargeInt = Int64
```

Large (64-bit) integer.

```
PBookmarkFlag = ^TBookmarkFlag
```

PBookmarkFlag is a convenience type, defined for internal use in TDataset (496) or one of it's descendents.

```
PBufferList = ^TBufferList
```

PBufferList is a pointer to a structure of type TBufferList (441). It is an internal type, and should not be used in end-user code.

```
PDateTimeRec = ^TdateTimeRec
```

Pointer to TDateTimeRec record.

```
PLargeInt = ^LargeInt
```

Pointer to Large (64-bit) integer.

```
PLookupListRec = ^TLookupListRec
```

Pointer to TLookupListRec record.

```
TBlobData = TBytes
```

TBlobData should never be used directly in application code.

```
TBlobStreamMode = (bmRead,bmWrite,bmReadWrite)
```

Table 16.2: Enumeration values for type TBlobStreamMode

Value	Explanation
bmRead	Read blob data.
bmReadWrite	Read and write blob data.
bmWrite	Write blob data.

`TBlobStramMode` is used when creating a stream for reading BLOB data. It indicates what the data will be used for: reading, writing or both.

```
TBlobType = ftBlob..ftWideMemo deprecated
```

`TBlobType` is a subrange type, indicating the various datatypes of BLOB fields.

```
TBookMark = TBytes
```

`TBookMark` is the type used by the TDataset.SetBookMark (496) method. It is of type `TBytes`, the contents of the buffer is determined by the actual `TDataset` descendent.

```
TBookmarkFlag = (bfCurrent,bfBOF,bfEOF,bfInserted)
```

Table 16.3: Enumeration values for type TBookmarkFlag

Value	Explanation
bfBOF	First record in the dataset.
bfCurrent	Buffer used for the current record.
bfEOF	Last record in the dataset.
bfInserted	Buffer used for insert.

`TBookmarkFlag` is used internally by TDataset (496) and it's descendent types to mark the internal memory buffers. It should not be used in end-user applications.

```
TBookmarkStr = ansistring
```

`TBookmarkStr` was the type used by the TDataset.Bookmark (518) property in earlier versions of FPC. It can be used as a string, but should in fact be considered an opaque type.

This type is deprecated, and no longer usable, use the TBookmark (441) type instead.

```
TBufferArray = ^TRecordBuffer
```

`TBufferArray` is an internally used type. It can change in future implementations, and should not be used in application code.

```
TBufferList = Array[0..dsMaxBufferCount-1] of TRecordBuffer
```

`TBufferList` is used internally by the TDataset (496) class to manage the memory buffers for the data. It should not be necessary to use this type in end-user applications.

```
TCloseErrorEvent = procedure(Sender: TObject; aError: Exception)
  of object
```

```
TDataAction = (daFail,daAbort,daRetry)
```

Table 16.4: Enumeration values for type TDataAction

Value	Explanation
daAbort	The operation should be aborted (edits are undone, and an EAbort exception is raised).
daFail	The operation should fail (an exception will be raised).
daRetry	Retry the operation.

`TDataAction` is used by the TDataSetErrorEvent (443) event handler prototype. The parameter `Action` of this event handler is of `TDataAction` type, and should indicate what action must be taken by the dataset.

```
TDatabaseClass = Class of TDataBase
```

`TDatabaseClass` is the class pointer for the TDatabase (487) class.

```
TDataChangeEvent = procedure(Sender: TObject; Field: TField) of
  object
```

`TDataChangeEvent` is the event handler prototype for the TDatasource.OnDataChange (539) event. The sender parameter is the `TDatasource` instance that triggered the event, and the `Field` parameter is the field whose data has changed. If the dataset has scrolled, then the `Field` parameter is `Nil`.

```
TDataEvent = (deFieldChange,deRecordChange,deDataSetChange,
  deDataSetScroll,deLayoutChange,deUpdateRecord,
  deUpdateState,deCheckBrowseMode,dePropertyChange,
  deFieldListChange,deFocusControl,deParentScroll,
  deConnectChange,deReconcileError,deDisabledStateChange)
```

Table 16.5: Enumeration values for type TDataEvent

Value	Explanation
deCheckBrowseMode	The browse mode is being checked.
deConnectChange	Unused.
deDataSetChange	The dataset property changed.
deDataSetScroll	The dataset scrolled to another record.
deDisabledStateChange	Unused.
deFieldChange	A field value changed.
deFieldListChange	Event sent when the list of fields of a dataset changes.
deFocusControl	Event sent whenever a control connected to a field should be focused.
deLayoutChange	The layout properties of one of the fields changed.
deParentScroll	Unused.
dePropertyChange	Unused.
deReconcileError	Unused.
deRecordChange	The current record changed.
deUpdateRecord	The record is being updated.
deUpdateState	The dataset state is updated.

`TDataEvent` describes the various events that can be sent to TDatasource (536) instances connected to a TDataset (496) instance.

```
TDataOperation = procedure of object
```

`TDataOperation` is a prototype handler used internally in `TDataset`. It can be changed at any time, so it should not be used in end-user code.

```
TDatasetClass = Class of TDataSet
```

`TDatasetClass` is the class type for the TDataset (496) class. It is currently unused in the DB unit and is defined for the benefit of other units.

```
TDataSetErrorEvent = procedure(DataSet: TDataSet; E: EDatabaseError
  ;
                                var DataAction: TDataAction) of
  object
```

`TDatasetErrorEvent` is used by the TDataset.OnEditError (533), TDataset.OnPostError (534) and TDataset.OnDeleteError (533) event handlers to allow the programmer to specify what should be done if an update operation fails with an exception: The `Dataset` parameter indicates what dataset triggered the event, the `E` parameter contains the exception object. The `DataAction` must be set by the event handler, and based on it's return value, the dataset instance will take appropriate action. The default value is `daFail`, i.e. the exception will be raised again. For a list of available return values, see TDataAction (442).

```
TDataSetNotifyEvent = procedure(DataSet: TDataSet) of object
```

`TDatasetNotifyEVent` is used in most of the TDataset (496) event handlers. It differs from the more general `TNotifyEvent` (defined in the Classes unit) in that the `Sender` parameter of the latter is replaced with the `Dataset` parameter. This avoids typecasts, the available TDataset methods can be used directly.

```
TDataSetState = (dsInactive,dsBrowse,dsEdit,dsInsert,dsSetKey,
  dsCalcFields,dsFilter,dsNewValue,dsOldValue,dsCurValue
  ,
                dsBlockRead,dsInternalCalc,dsOpening,dsRefreshFields
  )
```

Table 16.6: Enumeration values for type TDataSetState

Value	Explanation
dsBlockRead	The dataset is open, but no events are transferred to datasources.
dsBrowse	The dataset is active, and the cursor can be used to navigate the data.
dsCalcFields	The dataset is calculating it's calculated fields.
dsCurValue	The dataset is showing the current values of a record.
dsEdit	The dataset is in editing mode: the current record can be modified.
dsFilter	The dataset is filtering records.
dsInactive	The dataset is not active. No data is available.
dsInsert	The dataset is in insert mode: the current record is a new record which can be edited.
dsInternalCalc	The dataset is calculating it's internally calculated fields.
dsNewValue	The dataset is showing the new values of a record.
dsOldValue	The dataset is showing the old values of a record.
dsOpening	The dataset is currently opening, but is not yet completely open.
dsRefreshFields	Dataset is refreshing field values from server after an update.
dsSetKey	The dataset is calculating the primary key.

`TDataSetState` describes the current state of the dataset. During it's lifetime, the dataset's state is described by these enumerated values.

Some state are not used in the default TDataset implementation, and are only used by certain descendents.

```
TDateTimeAlias = TDateTime
```

`TDateTimeAlias` is no longer used.

```
TDateTimeRec = record
case TFieldType of
ftDate: (
  Date : LongInt
  ;
);
ftTime: (
  Time : LongInt;
);
ftDateTime: (
  DateTime : TDateTimeAlias
  ;
);
end
```

`TDateTimeRec` was used by older TDataset (496) implementations to store date/time values. Newer implementations use the `TDateTime`. This type should no longer be used.

```
TDBDatasetClass = Class of TDBDataset
```

TDBDatasetClass is the class pointer for TDBDataset (542)

```
TFieldAttribute = (faHiddenCol,faReadonly,faRequired,faLink,faUnNamed
  ,
                   faFixed)
```

Table 16.7: Enumeration values for type TFieldAttribute

Value	Explanation
faFixed	Fixed length field.
faHiddenCol	Field is a hidden column (used to construct a unique key).
faLink	Field is a link field for other datasets.
faReadonly	Field is read-only.
faRequired	Field is required.
faUnNamed	Field has no original name.

TFieldAttribute is used to denote some attributes of a field in a database. It is used in the Attributes (579) property of TFieldDef (575).

```
TFieldAttributes = Set of TFieldAttribute
```

TFieldAttributes is used in the TFieldDef.Attributes (579) property to denote additional attributes of the underlying field.

```
TFieldChars = Set of Char
```

TFieldChars is a type used in the TField.ValidChars (566) property. It's a simple set of characters.

```
TFieldClass = Class of TField
```

```
TFieldDefClass = Class of TFieldDef
```

TFieldDefClass is used to be able to customize the actual TDataset.FieldDefs (521) items class.

```
TFieldDefsClass = Class of TFieldDefs
```

TFieldDefClass is used to be able to customize the actual TDataset.FieldDefs (521) class used in a TDataset (496) descendent.

```
TFieldGetTextEvent = procedure(Sender: TField; var aText: string;
  DisplayText: Boolean) of object
```

TFieldGetTextEvent is the prototype for the TField.OnGetText (574) event handler. It should be used when the text of a field requires special formatting. The event handler should return the contents of the field in formatted form in the AText parameter. The DisplayText is True if the text is used for displaying purposes or is False if it will be used for editing purposes.

```
TFieldKind = (fkData,fkCalculated,fkLookup,fkInternalCalc)
```

Table 16.8: Enumeration values for type TFieldKind

Value	Explanation
fkCalculated	The field is calculated on the fly.
fkData	Field represents actual data in the underlying data structure.
fkInternalCalc	Field is calculated but stored in an underlying buffer.
fkLookup	The field is a lookup field.

`TFieldKind` indicates the type of a `TField` instance. Besides `TField` instances that represent fields present in the underlying data records, there can also be calculated or lookup fields. To distinguish between these kind of fields, `TFieldKind` is introduced.

```
TFieldKinds = Set of TFieldKind
```

`TFieldKinds` is a set of TFieldKind (446) values. It is used internally by the classes of the DB unit.

```
TFieldMap = Array[TFieldType] of Byte
```

`TFieldMap` is no longer used.

```
TFieldNotifyEvent = procedure(Sender: TField) of object
```

`TFieldNotifyEvent` is a prototype for the event handlers in the TField (549) class. It's `Sender` parameter is the field instance that triggered the event.

```
TFieldRef = ^TField
```

Pointer to a TField instance.

```
TFieldsClass = Class of TFields
```

`TFieldsClass` is needed to be able to specify the class of fields used in TDataset.Fields (524);

```
TFieldSetTextEvent = procedure(Sender: TField; const aText: string
  )
                                  of object
```

`TFieldSetTextEvent` is the prototype for an event handler used to set the contents of a field based on a user-edited text. It should be used when the text of a field is entered with special formatting. The event handler should set the contents of the field based on the formatted text in the `AText` parameter.

```
TFieldType = (ftUnknown,ftString,ftSmallint,ftInteger,ftWord,ftBoolean
  ,
              ftFloat,ftCurrency,ftBCD,ftDate,ftTime,ftDateTime
  ,ftBytes,
```

```
              ftVarBytes,ftAutoInc,ftBlob,ftMemo,ftGraphic
,ftFmtMemo,
              ftParadoxOle,ftDBaseOle,ftTypedBinary,ftCursor
,
              ftFixedChar,ftWideString,ftLargeint,ftADT,ftArray
,
              ftReference,ftDataSet,ftOraBlob,ftOraClob,ftVariant
,
              ftInterface,ftIDispatch,ftGuid,ftTimeStamp,ftFMTBcd
,
              ftFixedWideChar,ftWideMemo,ftOraTimeStamp,ftOraInterval
,
              ftLongWord,ftShortint,ftByte,ftExtended,ftSingle)
```

Table 16.9: Enumeration values for type TFieldType

Value	Explanation
ftADT	ADT value.
ftArray	Array data.
ftAutoInc	Auto-increment integer value (4 bytes).
ftBCD	Binary Coded Decimal value (DECIMAL and NUMERIC SQL types).
ftBlob	Binary data value (no type, no size).
ftBoolean	Boolean value.
ftByte	Byte field type.
ftBytes	Array of bytes value, fixed size (unytped).
ftCurrency	Currency value (4 decimal points).
ftCursor	Cursor data value (no size).
ftDataSet	Dataset data (blob).
ftDate	Date value.
ftDateTime	Date/Time (timestamp) value.
ftDBaseOle	Paradox OLE field data.
ftExtended	Extended field type.
ftFixedChar	Fixed character array (string).
ftFixedWideChar	Fixed wide character date (2 bytes per character).
ftFloat	Floating point value (double).
ftFMTBcd	Formatted BCD (Binary Coded Decimal) value.
ftFmtMemo	Formatted memo data value (no size).
ftGraphic	Graphical data value (no size).
ftGuid	GUID data value.
ftIDispatch	Dispatch data value.
ftInteger	Regular integer value (4 bytes, signed).
ftInterface	interface data value.
ftLargeint	Large integer value (8-byte).
ftLongWord	Longword (cardinal) field type.
ftMemo	Binary text data (no size).
ftOraBlob	Oracle BLOB data.
ftOraClob	Oracle CLOB data.
ftOraInterval	Oracle interval field type.
ftOraTimeStamp	Oracle time stamp field type.
ftParadoxOle	Paradox OLE field data (no size).
ftReference	Reference data.
ftShortint	Shortint field type.
ftSingle	
ftSmallint	Small integer value(1 byte, signed).
ftString	String data value (ansistring).
ftTime	Time value.
ftTimeStamp	Timestamp data value.
ftTypedBinary	Binary typed data (no size).
ftUnknown	Unknown data type.
ftVarBytes	Array of bytes value, variable size (untyped).
ftVariant	Variant data value.
ftWideMemo	Widestring memo data.
ftWideString	Widestring (2 bytes per character).
ftWord	Word-sized value(2 bytes, unsigned).

`TFieldType` indicates the type of a TField (549) underlying data, in the DataType (562) property.

```
TFilterOption = (foCaseInsensitive,foNoPartialCompare)
```

Table 16.10: Enumeration values for type TFilterOption

Value	Explanation
foCaseInsensitive	Filter case insensitively.
foNoPartialCompare	Do not compare values partially, always compare completely.

`TFilterOption` enumerates the various options available when filtering a dataset. The TFilterOptions (449) set is used in the TDataset.FilterOptions (526) property to indicate which of the options should be used when filtering the data.

```
TFilterOptions = Set of TFilterOption
```

`TFilterOption` is the set of filter options to use when filtering a dataset. This set type is used in the TDataset.FilterOptions (526) property. The available values are described in the TFilterOption (449) type.

```
TFilterRecordEvent = procedure(DataSet: TDataSet; var Accept: Boolean
  )
                                    of object
```

`TFilterRecordEvent` is the prototype for the TDataset.OnFilterRecord (534) event handler. The `Dataset` parameter indicates which dataset triggered the event, and the `Accept` parameter must be set to `true` if the current record should be shown, `False` should be used when the record should be hidden.

```
TGetMode = (gmCurrent,gmNext,gmPrior)
```

Table 16.11: Enumeration values for type TGetMode

Value	Explanation
gmCurrent	Retrieve the current record.
gmNext	Retrieve the next record.
gmPrior	Retrieve the previous record.

`TGetMode` is used internally by TDataset (496) when it needs to fetch more data for its buffers (using `GetRecord`). It tells the descendent dataset what operation must be performed.

```
TGetResult = (grOK,grBOF,grEOF,grError)
```

Table 16.12: Enumeration values for type TGetResult

Value	Explanation
grBOF	The beginning of the recordset is reached.
grEOF	The end of the recordset is reached.
grError	An error occurred.
grOK	The operation was completed successfully.

TGetResult is used by descendents of TDataset (496) when they have to communicate the result of the GetRecord operation back to the TDataset record.

```
TIndexOption = (ixPrimary,ixUnique,ixDescending,ixCaseInsensitive
  ,
                ixExpression,ixNonMaintained)
```

Table 16.13: Enumeration values for type TIndexOption

Value	Explanation
ixCaseInsensitive	The values in the index are sorted case-insensitively.
ixDescending	The values in the index are sorted descending.
ixExpression	The values in the index are based on a calculated expression.
ixNonMaintained	The index is non-maintained, i.e. changing the data will not update the index.
ixPrimary	The index is the primary index for the data.
ixUnique	The index is a unique index, i.e. each index value can occur only once.

TIndexOption describes the various properties that an index can have. It is used in the TIndexOptions (450) set type to describe all properties of an index definition as in TIndexDef (596).

```
TIndexOptions = Set of TIndexOption
```

TIndexOptions contains the set of properties that an index can have. It is used in the TIndexDef.Options (598) property to describe all properties of an index definition as in TIndexDef (596).

```
TLocateOption = (loCaseInsensitive,loPartialKey)
```

Table 16.14: Enumeration values for type TLocateOption

Value	Explanation
loCaseInsensitive	Perform a case-insensitive search.
loPartialKey	Accept partial key matches for string fields.

TLocateOption is used in the TDataset.Locate (513) call to enumerate the possible options available when locating a record in the dataset.

For string-type fields, this option indicates that fields starting with the search value are considered a match. For other fields (e.g. integer, date/time), this option is ignored and only equal field values are considered a match.

```
TLocateOptions = Set of TLocateOption
```

TLocateOptions is used in the TDataset.Locate (513) call: It should contain the actual options to use when locating a record in the dataset.

```
TLoginEvent = procedure(Sender: TObject; Username: string;
  Password: string) of object
```

`TLoginEvent` is the prototype for the TCustomConnection.OnLogin (486) event handler. It gets passed the `TCustomConnection` instance that is trying to login, and the initial username and password.

```
TParamBinding = Array of Integer
```

`TParamBinding` is an auxiliary type used when parsing and binding parameters in SQL statements. It should never be used directly in application code.

```
TParamClass = Class of TParam
```

`TParamClass` is needed to be able to specify the type of parameters when instantiating a TParams (631) collection.

```
TParamStyle = (psInterbase,psPostgreSQL,psSimulated)
```

Table 16.15: Enumeration values for type TParamStyle

Value	Explanation
psInterbase	Parameters are specified by a ? character.
psPostgreSQL	Parameters are specified by a $N character.
psSimulated	Parameters are specified by a $N character.

`TParamStyle` denotes the style in which parameters are specified in a query. It is used in the TParams.ParseSQl (634) method, and can have the following values:

psInterbase Parameters are specified by a ? character.

psPostgreSQL Parameters are specified by a $N character.

psSimulated Parameters are specified by a $N character.

```
TParamType = (ptUnknown,ptInput,ptOutput,ptInputOutput,ptResult)
```

Table 16.16: Enumeration values for type TParamType

Value	Explanation
ptInput	Input parameter.
ptInputOutput	Input/output parameter.
ptOutput	Output parameter, filled on result.
ptResult	Result parameter.
ptUnknown	Unknown type.

`TParamType` indicates the kind of parameter represented by a TParam (617) instance. it has one of the following values:

ptUnknown Unknown type.

ptInput Input parameter.

ptOutput Output parameter, filled on result.

ptInputOutput Input/output parameter.

ptResult Result parameter.

```
TParamTypes = Set of TParamType
```

`TParamTypes` is defined for completeness: a set of TParamType (451) values.

```
TProviderFlag = (pfInUpdate,pfInWhere,pfInKey,pfHidden,
  pfRefreshOnInsert,pfRefreshOnUpdate)
```

Table 16.17: Enumeration values for type TProviderFlag

Value	Explanation
pfHidden	
pfInKey	Field is a key field and used in the WHERE clause of an update statement.
pfInUpdate	Changes to the field should be propagated to the database.
pfInWhere	Field should be used in the WHERE clause of an update statement in case of `upWhereChanged`.
pfRefreshOnInsert	This field's value should be refreshed after insert.
pfRefreshOnUpdate	This field's value should be refreshed after update.

`TProviderFlag` describes how the field should be used when applying updates from a dataset to the database. Each field of a TDataset (496) has one or more of these flags.

```
TProviderFlags = Set of TProviderFlag
```

`TProviderFlags` is used for the TField.ProviderFlags (573) property to describe the role of the field when applying updates to a database.

```
TPSCommandType = (ctUnknown,ctQuery,ctTable,ctStoredProc,ctSelect
  ,
                  ctInsert,ctUpdate,ctDelete,ctDDL)
```

Table 16.18: Enumeration values for type TPSCommandType

Value	Explanation
ctDDL	SQL DDL statement.
ctDelete	SQL DELETE Statement.
ctInsert	SQL INSERT Statement.
ctQuery	General SQL statement.
ctSelect	SQL SELECT Statement.
ctStoredProc	Stored procedure statement.
ctTable	Table contents (select * from table).
ctUnknown	Unknown SQL type or not SQL based.
ctUpdate	SQL UPDATE statement.

TPSCommandType is used in the IProviderSupport.PSGetCommandType (463) call to determine the type of SQL command that the provider is exposing. It is meaningless for datasets that are not SQL based.

```
TRecordBuffer = PAnsiChar
```

TRecordBuffer is the type used by TDataset (496) to point to a record's data buffer. It is used in several internal TDataset routines.

```
TRecordBufferBaseType = AnsiChar
```

TRecordBufferBaseType should not be used directly. It just serves as an (opaque) base type to TRecordBuffer (453)

```
TResolverResponse = (rrSkip,rrAbort,rrMerge,rrApply,rrIgnore)
```

Table 16.19: Enumeration values for type TResolverResponse

Value	Explanation
rrAbort	Abort the whole update process, no error message is displayed (no EAbort exception raised).
rrApply	Replace the update with new values applied by the event handler.
rrIgnore	Ignore the error and remove update from change log.
rrMerge	Merge the update with existing changes on the server.
rrSkip	Skip the current update, leave it in the change log.

TResolverResponse is used to indicate what should happen to a pending change that could not be resolved. It is used in callbacks.

```
TResyncMode= Set of (rmExact,rmCenter)
```

Table 16.20: Enumeration values for type

Value	Explanation
rmCenter	Try to position the cursor in the middle of the buffer.
rmExact	Reposition at exact the same location in the buffer.

TResyncMode is used internally by various TDataset (496) navigation and data manipulation methods such as the TDataset.Refresh (516) method when they need to reset the cursor position in the dataset's buffer.

```
TSQLParseOption = (spoCreate,spoEscapeSlash,spoEscapeRepeat,spoUseMacro
  )
```

Table 16.21: Enumeration values for type TSQLParseOption

Value	Explanation
spoCreate	Indicates existing parameters are cleared and re-created, not updated.
spoEscapeRepeat	Causes an escaped character to be repeated.
spoEscapeSlash	Causes the Slash character ('/') to be escaped.
spoUseMacro	Enables macro expansion in a SQL statement.

TSQLParseOption is an enumerated type with values that represent SQL parser options available for use in the `TParams` collection. Value(s) from TSQLParseOption are stored in the `TSQLParseOptions` set type, and passed as an argument to the `TParams.ParseSQL` method. When a value from the enumeration is included in the set, the feature or behavior is enabled while parsing the SQL statement.

```
TSQLParseOptions = Set of TSQLParseOption
```

`TSQLParseOptions` is a set type used to store zero or more values from the `TSQLParseOption` enumeration. TSQLParseOptions is used in the `TParams.ParseSQL` method to indicate the options enabled when the SQL statement is parsed. The set type can be passed as an argument to the method.

See TSQLParseOption (453) for information about the enumeration values and their meanings.

```
TStringFieldBuffer = Array[0..dsMaxStringSize] of AnsiChar
```

Type to access string field content buffers as an array of characters.

```
TUpdateAction = (uaFail,uaAbort,uaSkip,uaRetry,uaApplied)
```

Table 16.22: Enumeration values for type TUpdateAction

Value	Explanation
uaAbort	The whole update operation should abort.
uaApplied	Consider the update as applied.
uaFail	Update operation should fail.
uaRetry	Retry the update operation.
uaSkip	The update of the current record should be skipped. (but not discarded).

`TUpdateAction` indicates what action must be taken in case the applying of updates on the underlying database fails. This type is not used in the TDataset (496) class, but is defined on behalf of `TDataset` descendents that implement caching of updates: It indicates what should be done when the (delayed) applying of the updates fails. This event occurs long after the actual post or delete operation.

```
TUpdateKind = (ukModify,ukInsert,ukDelete)
```

Table 16.23: Enumeration values for type TUpdateKind

Value	Explanation
ukDelete	Delete a record in the database.
ukInsert	insert a new record in the database.
ukModify	Modify an existing record in the database.

`TUpdateKind` indicates what kind of update operation is in progress when applying updates.

```
TUpdateMode = (upWhereAll,upWhereChanged,upWhereKeyOnly)
```

Table 16.24: Enumeration values for type TUpdateMode

Value	Explanation
upWhereAll	Use all old field values.
upWhereChanged	Use only old field values of modified fields.
upWhereKeyOnly	Only use key fields in the where clause.

`TUpdateMode` determines how the `WHERE` clause of update queries for SQL databases should be constructed.

```
TUpdateStatus = (usUnmodified,usModified,usInserted,usDeleted)
```

Table 16.25: Enumeration values for type TUpdateStatus

Value	Explanation
usDeleted	Record exists in the database, but is locally deleted.
usInserted	Record does not yet exist in the database, but is locally inserted.
usModified	Record exists in the database but is locally modified.
usUnmodified	Record is unmodified.

`TUpdateStatus` determines the current state of the record buffer, if updates have not yet been applied to the database.

```
TUpdateStatusSet = Set of TUpdateStatus
```

`TUpdateStatusSet` is a set of TUpdateStatus (455) values.

16.3.3 Variables

```
LoginDialogExProc : function(const ADatabaseName: string; var AUserName
  : string;
        var APassword: string; UserNameReadOnly: Boolean
  ) : Boolean = Nil
```

`LoginDialogExProc` is a procedural variable that can be set to handle login dialogs: if a database connection component needs to collect login data (typically when `LoginPrompt` is `True`), then if this callback is set it can e.g. be used to show a dialog used to fetch the data.

16.4 Procedures and functions

16.4.1 BuffersEqual

Synopsis: Check whether 2 memory buffers are equal.

Declaration:
```
function BuffersEqual(Buf1: Pointer; Buf2: Pointer; Size: Integer)
                      : Boolean
```

Visibility: default

Description: `BuffersEqual` compares the memory areas pointed to by the `Buf1` and `Buf2` pointers and returns `True` if the contents are equal. The memory areas are compared for the first `Size` bytes. If all bytes in the indicated areas are equal, then `True` is returned, otherwise `False` is returned.

Errors: If `Buf1` or `Buf2` do not point to a valid memory area or `Size` is too large, then an exception may occur

See also: #rtl.sysutils.Comparemem (**??**)

16.4.2 DatabaseError

Synopsis: Raise an `EDatabaseError` exception.

Declaration:
```
procedure DatabaseError(const Msg: string);  Overload
procedure DatabaseError(const Msg: string; Comp: TComponent);  Overload
```

Visibility: default

Description: `DatabaseError` raises an EDatabaseError (458) exception, passing it `Msg`. If `Comp` is specified, the name of the component is prepended to the message.

See also: DatabaseErrorFmt (456), EDatabaseError (458)

16.4.3 DatabaseErrorFmt

Synopsis: Raise an `EDatabaseError` exception with a formatted message.

Declaration:
```
procedure DatabaseErrorFmt(const Fmt: string;
                           const Args: Array of const);  Overload
procedure DatabaseErrorFmt(const Fmt: string;
                           const Args: Array of const; Comp: TComponent)
                           ;  Overload
```

Visibility: default

Description: `DatabaseErrorFmt` raises an EDatabaseError (458) exception, passing it a message made by calling #rtl.sysutils.format (**??**) with the `fmt` and `Args` arguments. If `Comp` is specified, the name of the component is prepended to the message.

See also: DatabaseError (456), EDatabaseError (458)

16.4.4 DateTimeRecToDateTime

Synopsis: Convert `TDateTimeRec` record to a `TDateTime` value.

Declaration:
```
function DateTimeRecToDateTime(DT: TFieldType; Data: TDateTimeRec)
                                    : TDateTime
```

Visibility: default

Description: `DateTimeRecToDateTime` examines `Data` and `Dt` and uses `dt` to convert the timestamp in `Data` to a `TDateTime` value.

See also: TFieldType (447), TDateTimeRec (444), DateTimeToDateTimeRec (457)

16.4.5 DateTimeToDateTimeRec

Synopsis: Convert `TDateTime` value to a `TDateTimeRec` record.

Declaration:
```
function DateTimeToDateTimeRec(DT: TFieldType; Data: TDateTime)
                                    : TDateTimeRec
```

Visibility: default

Description: `DateTimeToDateTimeRec` examines `Data` and `Dt` and uses `dt` to convert the date/time value in `Data` to a `TDateTimeRec` record.

See also: TFieldType (447), TDateTimeRec (444), DateTimeRecToDateTime (457)

16.4.6 DisposeMem

Synopsis: Dispose of a heap memory block and `Nil` the pointer (deprecated).

Declaration:
```
procedure DisposeMem(var Buffer; Size: Integer)
```

Visibility: default

Description: `DisposeMem` disposes of the heap memory area pointed to by `Buffer` (`Buffer` must be of type `Pointer`). The `Size` parameter indicates the size of the memory area (it is, in fact, ignored by the heap manager). The pointer `Buffer` is set to `Nil`. If `Buffer` is `Nil`, then nothing happens. Do not use `DisposeMem` on objects, because their destructor will not be called.

Errors: If `Buffer` is not pointing to a valid heap memory block, then memory corruption may occur.

See also: #rtl.system.FreeMem (**??**), #rtl.sysutils.freeandnil (**??**)

16.4.7 enumerator(TDataSet):TDataSetEnumerator

Synopsis: Operator to return dataset enumerator.

Declaration:
```
operator enumerator(ADataSet: TDataSet) : TDataSetEnumerator
```

Visibility: default

Description: This operator allows to use the TDataSetEnumerator (535) as an enumerator for a TDataset (496)

See also: TDataSetEnumerator (535), TDataset (496)

16.4.8 ExtractFieldName

Synopsis: Extract the field name at position.

Declaration:
```
function ExtractFieldName(const Fields: string; var Pos: Integer)
                          : string
```

Visibility: default

Description: `ExtractFieldName` returns the string starting at position `Pos` till the next semicolon (;) character or the end of the string. On return, `Pos` contains the position of the first character after the semicolon character (or one more than the length of the string).

See also: Tfields.GetFieldList (584)

16.4.9 SkipComments

Synopsis: Skip SQL comments.

Declaration:
```
function SkipComments(var p: PChar; EscapeSlash: Boolean;
                      EscapeRepeat: Boolean) : Boolean
```

Visibility: default

Description: `SkipComments` examines the null-terminated string in `P` and skips any SQL comment or string literal found at the start. It returns `P` the first non-comment or non-string literal position. The `EscapeSlash` parameter determines whether the backslash character (\) functions as an escape character (i.e. the following character is not considered a delimiter). `EscapeRepeat` must be set to `True` if the quote character is repeated to indicate itself.

The function returns `True` if a comment was found and skipped, `False` otherwise.

Errors: No checks are done on the validity of `P`.

See also: TParams.ParseSQL (634)

16.5 TLookupListRec

```
TLookupListRec = record
  Key : Variant;
  Value : Variant;
end
```

`TLookupListRec` is used by lookup fields to store lookup results, if the results should be cached. Its two fields keep the key value and associated lookup value.

16.6 EDatabaseError

16.6.1 Description

`EDatabaseError` is the base class from which database-related exception classes should derive. It is raised by the DatabaseError (456) call.

See also: DatabaseError (456), DatabaseErrorFmt (456)

16.7 EUpdateError

16.7.1 Description

`EupdateError` is an exception used by the TProvider database support. It should never be raised directly.

See also: EDatabaseError (458)

16.7.2 Method overview

Page	Method	Description
459	Create	Create a new EUpdateError instance.
459	Destroy	Free the EupdateError instance.

16.7.3 Property overview

Page	Properties	Access	Description
460	Context	r	Context in which exception occurred.
460	ErrorCode	r	Numerical error code.
460	OriginalException	r	Originally raised exception.
460	PreviousError	r	Previous error number.

16.7.4 EUpdateError.Create

Synopsis: Create a new EUpdateError instance.

Declaration:
```
constructor Create(NativeError: string; Context: string;
                   ErrCode: Integer; PrevError: Integer; E: Exception)
```

Visibility: public

Description: `Create` instantiates a new `EUpdateError` object and populates the various properties with the `NativeError`, `Context`, `ErrCode` and `PrevError` parameters. The `E` parameter is the actual exception that occurred while the update operation was attempted. The exception object `E` will be freed if the `EUpdateError` instance is freed.

See also: EDatabaseError (458)

16.7.5 EUpdateError.Destroy

Synopsis: Free the EupdateError instance.

Declaration:
```
destructor Destroy;  Override
```

Visibility: public

Description: `Destroy` frees the original exception object (if there was one) and then calls the inherited destructor.

Errors: If the original exception object was already freed, an error will occur.

See also: EUpdateError.OriginalException (460)

16.7.6 EUpdateError.Context

Synopsis: Context in which exception occurred.

Declaration: `Property Context : string`

Visibility: public

Access: Read

Description: A description of the context in which the original exception was raised.

See also: EUpdateError.OriginalException (460), EUpdateError.ErrorCode (460), EUpdateError.PreviousError (460)

16.7.7 EUpdateError.ErrorCode

Synopsis: Numerical error code.

Declaration: `Property ErrorCode : Integer`

Visibility: public

Access: Read

Description: `ErrorCode` is a numerical error code, provided by the native data access layer, to describe the error. It may or not be filled.

See also: EUpdateError.OriginalException (460), EUpdateError.Context (460), EUpdateError.PreviousError (460)

16.7.8 EUpdateError.OriginalException

Synopsis: Originally raised exception.

Declaration: `Property OriginalException : Exception`

Visibility: public

Access: Read

Description: `OriginalException` is the originally raised exception that is transformed to an `EUpdateError` exception.

See also: DB.EDatabaseError (458)

16.7.9 EUpdateError.PreviousError

Synopsis: Previous error number.

Declaration: `Property PreviousError : Integer`

Visibility: public

Access: Read

Description: `PreviousError` is used to order the errors which occurred during an update operation.

See also: EUpdateError.ErrorCode (460), EUpdateError.Context (460), EUpdateError.OriginalException (460)

16.8 IProviderSupport

16.8.1 Description

`IProviderSupport` is an interface used by Delphi's TProvider (datasnap) technology. It is currently not used in Free Pascal, but is provided for Delphi compatibility. The TDataset (496) class implements all the methods of this interface for the benefit of descendent classes, but does not publish the interface in it's declaration.

See also: TDataset (496)

16.8.2 Method overview

Page	Method	Description
461	PSEndTransaction	End an active transaction.
461	PSExecute	Execute the current command-text.
462	PSExecuteStatement	Execute a SQL statement.
462	PSGetAttributes	Get a list of attributes (metadata).
462	PSGetCommandText	Return the SQL command executed for getting data.
463	PSGetCommandType	Return SQL command type.
463	PSGetDefaultOrder	Default order index definition.
463	PSGetIndexDefs	Return a list of index definitions.
463	PSGetKeyFields	Return a list of key fields in the dataset.
464	PSGetParams	Get the parameters in the commandtext.
464	PSGetQuoteChar	Quote character for quoted strings.
464	PSGetTableName	Name of database table which must be updated.
464	PSGetUpdateException	Transform exception to `UpdateError`.
465	PSInTransaction	Is the dataset in an active transaction.
465	PSIsSQLBased	Is the dataset SQL based.
465	PSIsSQLSupported	Can the dataset support SQL statements.
465	PSReset	Position the dataset on the first record.
466	PSSetCommandText	Set the command-text of the dataset.
466	PSSetParams	Set the parameters for the command text.
466	PSStartTransaction	Start a new transaction.
466	PSUpdateRecord	Update a record.

16.8.3 IProviderSupport.PSEndTransaction

Synopsis: End an active transaction.

Declaration:
```
procedure PSEndTransaction(ACommit: Boolean)
```

Visibility: default

Description: `PSEndTransaction` ends an active transaction if an transaction is active. (PSInTransaction (438) returns `True`). If `ACommit` is `True` then the transaction is committed, else it is rolled back.

See also: PSInTransaction (438), PSStartTransaction (438)

16.8.4 IProviderSupport.PSExecute

Synopsis: Execute the current command-text.

Declaration:
```
procedure PSExecute
```

Visibility: default

Description: `PSExecute` executes the current SQL statement: the command as it is returned by PSGetCommandText (438).

See also: PSGetCommandText (438), PSExecuteStatement (438)

16.8.5 IProviderSupport.PSExecuteStatement

Synopsis: Execute a SQL statement.

Declaration:
```
function PSExecuteStatement(const ASQL: string; AParams: TParams;
                            ResultSet: Pointer) : Integer
```

Visibility: default

Description: `PSExecuteStatement` will execute the `ASQL` SQL statement in the current transaction. The SQL statement can have parameters embedded in it (in the form `:ParamName`), values for these parameters will be taken from `AParams`. If the SQL statement returns a result-set, then the result set can be returned in `ResultSet`. The function returns `True` if the statement was executed successful.

`PSExecuteStatement` does not modify the content of `CommandText`: PSGetCommandText (438) returns the same value before and after a call to `PSExecuteStatement`.

See also: PSGetCommandText (438), PSSetCommandText (438), PSExecuteStatement (438)

16.8.6 IProviderSupport.PSGetAttributes

Synopsis: Get a list of attributes (metadata).

Declaration:
```
procedure PSGetAttributes(List: TList)
```

Visibility: default

Description: `PSGetAttributes` returns a set of `name=value` pairs which is included in the data packet sent to a client.

See also: PSGetCommandText (438)

16.8.7 IProviderSupport.PSGetCommandText

Synopsis: Return the SQL command executed for getting data.

Declaration:
```
function PSGetCommandText : string
```

Visibility: default

Description: `PSGetCommandText` returns the SQL command that is executed when the PSExecute (438) function is called (for a TSQLQuery this would be the SQL property) or when the dataset is opened.

See also: PSExecute (438), PSSetCommandText (438)

16.8.8 IProviderSupport.PSGetCommandType

Synopsis: Return SQL command type.

Declaration: `function PSGetCommandType : TPSCommandType`

Visibility: default

Description: `PSGetCommandType` should return the kind of SQL statement that is executed by the command (as returned by PSGetCommandText (438)). The list of possible command types is enumerated in TPSCommandType (452).

See also: PSGetCommandText (438), TPSCommandType (452), PSExecute (438)

16.8.9 IProviderSupport.PSGetDefaultOrder

Synopsis: Default order index definition.

Declaration: `function PSGetDefaultOrder : TIndexDef`

Visibility: default

Description: `PSGetDefaultOrder` should return the index definition from the list of indexes (as returned by PSGetIndexDefs (438)) that represents the default sort order.

See also: PSGetIndexDefs (438), PSGetKeyFields (438)

16.8.10 IProviderSupport.PSGetIndexDefs

Synopsis: Return a list of index definitions.

Declaration: `function PSGetIndexDefs(IndexTypes: TIndexOptions) : TIndexDefs`

Visibility: default

Description: `PSGetIndexDefs` should return a list of index definitions, limited to the types of indexes in `IndexTypes`.

See also: PSGetDefaultOrder (438), PSGetKeyFields (438)

16.8.11 IProviderSupport.PSGetKeyFields

Synopsis: Return a list of key fields in the dataset.

Declaration: `function PSGetKeyFields : string`

Visibility: default

Description: `PSGetKeyFields` returns a semicolon-separated list of fieldnames that make up the unique key for a record. Normally, these are the names of the fields that have `pfInKey` in their ProviderOptions (549) property.

See also: PSGetIndexDefs (438), PSGetDefaultOrder (438), TField.ProviderOptions (549), TProviderFlags (452)

16.8.12 IProviderSupport.PSGetParams

Synopsis: Get the parameters in the commandtext.

Declaration:
```
function PSGetParams : TParams
```

Visibility: default

Description: `PSGetParams` returns the list of parameters in the command-text (as returned by PSGetCommandText (438)). This is usually the `Params` property of a TDataset (496) descendant.

See also: PSGetCommandText (438), PSSetParams (438)

16.8.13 IProviderSupport.PSGetQuoteChar

Synopsis: Quote character for quoted strings.

Declaration:
```
function PSGetQuoteChar : string
```

Visibility: default

Description: `PSGetQuoteChar` returns the quote character needed to enclose string literals in an SQL statement for the underlying database.

See also: PSGetTableName (438)

16.8.14 IProviderSupport.PSGetTableName

Synopsis: Name of database table which must be updated.

Declaration:
```
function PSGetTableName : string
```

Visibility: default

Description: `PSGetTableName` returns the name of the table for which update SQL statements must be constructed. The provider can create and execute SQL statements to update the underlying database by itself. For this, it uses `PSGetTableName` as the name of the table to update.

See also: PSGetQuoteChar (438)

16.8.15 IProviderSupport.PSGetUpdateException

Synopsis: Transform exception to `UpdateError`.

Declaration:
```
function PSGetUpdateException(E: Exception; Prev: EUpdateError)
                                        : EUpdateError
```

Visibility: default

Description: `PSGetUpdateException` is called to transform and chain exceptions that occur during an ApplyUpdates operation. The exception `E` must be transformed to an EUpdateError (459) exception. The previous `EUpdateError` exception in the update batch is passed in `Prev`.

See also: EUpdateError (459)

16.8.16 IProviderSupport.PSInTransaction

Synopsis: Is the dataset in an active transaction.

Declaration: `function PSInTransaction : Boolean`

Visibility: default

Description: `PSInTransaction` returns `True` if the dataset is in an active transaction or `False` if no transaction is active.

See also: PSEndTransaction (438), PSStartTransaction (438)

16.8.17 IProviderSupport.PSIsSQLBased

Synopsis: Is the dataset SQL based.

Declaration: `function PSIsSQLBased : Boolean`

Visibility: default

Description: `PSIsSQLBased` returns `True` if the dataset is SQL based or not. Note that this is different from PSIsSQLSupported (438) which indicates whether SQL statements can be executed using PSExecuteCommand (438)

See also: PSIsSQLSupported (438), PSExecuteCommand (438)

16.8.18 IProviderSupport.PSIsSQLSupported

Synopsis: Can the dataset support SQL statements.

Declaration: `function PSIsSQLSupported : Boolean`

Visibility: default

Description: `PSIsSQLSupported` returns `True` if PSExecuteCommand (438) can be used to execute SQL statements on the underlying database.

See also: PSExecuteCommand (438)

16.8.19 IProviderSupport.PSReset

Synopsis: Position the dataset on the first record.

Declaration: `procedure PSReset`

Visibility: default

Description: `PSReset` repositions the dataset on the first record. For bi-directional datasets, this usually means that first is called, but for unidirectional datasets this may result in re-fetching the data from the underlying database.

See also: TDataset.First (510), TDataset.Open (515)

16.8.20 IProviderSupport.PSSetCommandText

Synopsis: Set the command-text of the dataset.

Declaration:
```
procedure PSSetCommandText(const CommandText: string)
```

Visibility: default

Description: `PSSetCommandText` sets the commandtext (SQL) statement that is executed by `PSExecute` or that is used to open the dataset.

See also: PSExecute (438), PSGetCommandText (438), PSSetParams (438)

16.8.21 IProviderSupport.PSSetParams

Synopsis: Set the parameters for the command text.

Declaration:
```
procedure PSSetParams(AParams: TParams)
```

Visibility: default

Description: `PSSetParams` sets the values of the parameters that should be used when executing the command-text SQL statement.

See also: PSSetCommandText (438), PSGetParams (438)

16.8.22 IProviderSupport.PSStartTransaction

Synopsis: Start a new transaction.

Declaration:
```
procedure PSStartTransaction
```

Visibility: default

Description: `PSStartTransaction` is used by the provider to start a new transaction. It will only be called if no transaction was active yet (i.e. PSIntransaction (438) returned `False`).

See also: PSEndTransaction (438), PSIntransaction (438)

16.8.23 IProviderSupport.PSUpdateRecord

Synopsis: Update a record.

Declaration:
```
function PSUpdateRecord(UpdateKind: TUpdateKind; Delta: TDataSet)
                                : Boolean
```

Visibility: default

Description: `PSUpdateRecord` is called before attempting to update the records through generated SQL statements. The update to be performed is passed in `UpdateKind` parameter. The `Delta` Dataset's current record contains all data for the record that must be updated.

The function returns `True` if the update was successfully applied, `False` if not. In that case the provider will attempt to update the record using SQL statements if the dataset allows it.

See also: PSIsSQLSupported (438), PSExecuteCommand (438)

16.9 TArrayField

16.9.1 Method overview

Page	Method	Description
467	Create	

16.9.2 TArrayField.Create

Declaration: `constructor Create(AOwner: TComponent); Override`

Visibility: public

16.10 TAutoIncField

16.10.1 Description

`TAutoIncField` is the class created when a dataset must manage 32-bit signed integer data, of datatype `ftAutoInc`: This field gets it's data automatically by the database engine. It exposes no new properties, but simply overrides some methods to manage 32-bit signed integer data.

It should never be necessary to create an instance of `TAutoIncField` manually, a field of this class will be instantiated automatically for each auto-incremental field when a dataset is opened.

See also: TField (549)

16.10.2 Method overview

Page	Method	Description
467	Create	Create a new instance of the `TAutoIncField` class.

16.10.3 TAutoIncField.Create

Synopsis: Create a new instance of the `TAutoIncField` class.

Declaration: `constructor Create(AOwner: TComponent); Override`

Visibility: public

Description: `Create` initializes a new instance of the `TAutoIncField` class. It simply calls the inherited constructor and then sets up some of the TField (549) class' fields.

See also: TField (549)

16.11 TBCDField

16.11.1 Description

`TBCDField` is the class used when a dataset must manage data of Binary Coded Decimal type. (TField.DataType (562) equals `ftBCD`). It initializes some of the properties of the TField (549) class, and overrides some of its methods to be able to work with BCD fields.

TBCDField assumes that the field's contents can be stored in a currency type, i.e. the maximum number of decimals after the decimal separator that can be stored in a TBCDField is 4. Fields that need to store a larger amount of decimals should be represented by a TFMTBCDField (592) instance.

It should never be necessary to create an instance of TBCDField manually, a field of this class will be instantiated automatically for each BCD field when a dataset is opened.

See also: TDataset (496), TField (549), TFMTBCDField (592)

16.11.2 Method overview

Page	Method	Description
468	CheckRange	Check whether a values falls within the allowed range.
468	Create	Create a new instance of a TBCDField class.

16.11.3 Property overview

Page	Properties	Access	Description
469	Currency	rw	Does the field represent a currency amount.
469	MaxValue	rw	Maximum value for the field.
470	MinValue	rw	Minimum value for the field.
469	Precision	rw	Precision of the BCD field.
470	Size		Number of decimals after the decimal separator.
469	Value	rw	Value of the field contents as a Currency type.

16.11.4 TBCDField.Create

Synopsis: Create a new instance of a TBCDField class.

Declaration:
```
constructor Create(AOwner: TComponent);  Override
```

Visibility: public

Description: Create initializes a new instance of the TBCDField class. It calls the inherited destructor, and then sets some TField (549) properties to configure the instance for working with BCD data values.

See also: TField (549)

16.11.5 TBCDField.CheckRange

Synopsis: Check whether a values falls within the allowed range.

Declaration:
```
function CheckRange(AValue: Currency) : Boolean
```

Visibility: public

Description: CheckRange returns True if AValue lies within the range defined by the MinValue (470) and MaxValue (469) properties. If the value lies outside of the allowed range, then False is returned.

See also: MaxValue (469), MinValue (470)

16.11.6 TBCDField.Value

Synopsis: Value of the field contents as a Currency type.

Declaration: `Property Value : Currency`

Visibility: public

Access: Read,Write

Description: `Value` is overridden from the TField.Value (566) property to a currency type field. It returns the same value as the TField.AsCurrency (556) field.

See also: TField.Value (566), TField.AsCurrency (556)

16.11.7 TBCDField.Precision

Synopsis: Precision of the BCD field.

Declaration: `Property Precision : LongInt`

Visibility: published

Access: Read,Write

Description: `Precision` is the total number of decimals in the BCD value. It is not the same as TBCDField.Size (470), which is the number of decimals after the decimal point. The `Precision` property should be set by the descendent classes when they initialize the field, and should be considered read-only. Changing the value will influence the values returned by the various `AsXXX` properties.

See also: TBCDField.Size (470), TBCDField.Value (469)

16.11.8 TBCDField.Currency

Synopsis: Does the field represent a currency amount.

Declaration: `Property Currency : Boolean`

Visibility: published

Access: Read,Write

Description: `Currency` can be set to `True` to indicate that the field contains data representing an amount of currency. This affects the way the TField.DisplayText (563) and TField.Text (565) properties format the value of the field: if the `Currency` property is `True`, then these properties will format the value as a currency value (generally appending the currency sign) and if the `Currency` property is `False`, then they will format it as a normal floating-point value.

See also: TField.DisplayText (563), TField.Text (565)

16.11.9 TBCDField.MaxValue

Synopsis: Maximum value for the field.

Declaration: `Property MaxValue : Currency`

Visibility: published

Access: Read,Write

Description: `MaxValue` can be set to a value different from zero, it is then the maximum value for the field if set to any value different from zero. When setting the field's value, the value may not be larger than `MaxValue`. Any attempt to write a larger value as the field's content will result in an exception. By default `MaxValue` equals `0`, i.e. any floating-point value is allowed.

If `MaxValue` is set, MinValue (470) should also be set, because it will also be checked.

See also: TBCDField.MinValue (470), TBCDField.CheckRange (468)

16.11.10 TBCDField.MinValue

Synopsis: Minimum value for the field.

Declaration: `Property MinValue : Currency`

Visibility: published

Access: Read,Write

Description: `MinValue` can be set to a value different from zero, then it is the minimum value for the field. When setting the field's value, the value may not be less than `MinValue`. Any attempt to write a smaller value as the field's content will result in an exception. By default `MinValue` equals `0`, i.e. any floating-point value is allowed.

If `MinValue` is set, TBCDField.MaxValue (469) should also be set, because it will also be checked.

See also: TBCDField.MaxValue (469), TBCDField.CheckRange (468)

16.11.11 TBCDField.Size

Synopsis: Number of decimals after the decimal separator.

Declaration: `Property Size :`

Visibility: published

Access:

Description: `Size` is the number of decimals after the decimal separator. It is not the total number of decimals, which is stored in the TBCDField.Precision (469) field.

See also: TBCDField.Precision (469)

16.12 TBinaryField

16.12.1 Description

`TBinaryField` is an abstract class, designed to handle binary data of variable size. It overrides some of the properties and methods of the TField (549) class to be able to work with binary field data, such as retrieving the contents as a string or as a variant.

One must never create an instance of `TBinaryField` manually, it is an abstract class. Instead, a descendent class such as TBytesField (478) or TVarBytesField (644) should be created.

See also: TDataset (496), TField (549), TBytesField (478), TVarBytesField (644)

16.12.2 Method overview

Page	Method	Description
471	Create	Create a new instance of a `TBinaryField` class.

16.12.3 Property overview

Page	Properties	Access	Description
471	Size		Size of the binary data.

16.12.4 TBinaryField.Create

Synopsis: Create a new instance of a `TBinaryField` class.

Declaration: `constructor Create(AOwner: TComponent); Override`

Visibility: public

Description: `Create` initializes a new instance of the `TBinaryField` class. It simply calls the inherited destructor.

See also: TField (549)

16.12.5 TBinaryField.Size

Synopsis: Size of the binary data.

Declaration: `Property Size :`

Visibility: published

Access:

Description: `Size` is simply redeclared published with a default value of 16.

See also: TField.Size (565)

16.13 TBlobField

16.13.1 Description

`TBLobField` is the class used when a dataset must manage BLOB data. (TField.DataType (562) equals `ftBLOB`). It initializes some of the properties of the TField (549) class, and overrides some of its methods to be able to work with BLOB fields. It also serves as parent class for some specialized blob-like field types such as TMemoField (612), TWideMemoField (645) or TGraphicField (595)

It should never be necessary to create an instance of `TBlobField` manually, a field of this class will be instantiated automatically for each BLOB field when a dataset is opened.

See also: TDataset (496), TField (549), TMemoField (612), TWideMemoField (645), TGraphicField (595)

16.13.2 Method overview

Page	Method	Description
472	Clear	Clear the BLOB field's contents.
472	Create	Create a new instance of a `TBlobField` class.
472	IsBlob	Is the field a blob field.
473	LoadFromFile	Load the contents of the field from a file.
473	LoadFromStream	Load the field's contents from stream.
473	SaveToFile	Save field contents to a file.
474	SaveToStream	Save the field's contents to stream.
474	SetFieldType	Set field type.

16.13.3 Property overview

Page	Properties	Access	Description
474	BlobSize	r	Size of the current blob.
475	BlobType	rw	Type of blob.
474	Modified	rw	Has the field's contents been modified.
475	Size		Size of the blob field.
475	Transliterate	rw	Should the contents of the field be transliterated.
475	Value	rw	Return the field's contents as a string.

16.13.4 TBlobField.Create

Synopsis: Create a new instance of a `TBlobField` class.

Declaration:
```
constructor Create(AOwner: TComponent);  Override
```

Visibility: public

Description: `Create` initializes a new instance of the `TBlobField` class. It calls the inherited destructor, and then sets some TField (549) properties to configure the instance for working with BLOB data.

See also: TField (549)

16.13.5 TBlobField.Clear

Synopsis: Clear the BLOB field's contents.

Declaration:
```
procedure Clear;  Override
```

Visibility: public

Description: `Clear` overrides the `TField` implementation of TField.Clear (553). It creates and immediately releases an empty blob stream in write mode, effectively clearing the contents of the BLOB field.

See also: TField.Clear (553), TField.IsNull (564)

16.13.6 TBlobField.IsBlob

Synopsis: Is the field a blob field.

Declaration:
```
class function IsBlob : Boolean;  Override
```

Visibility: public

Description: `IsBlob` is overridden by `TBlobField` to return `True`

See also: TField.IsBlob (554)

16.13.7 TBlobField.LoadFromFile

Synopsis: Load the contents of the field from a file.

Declaration: `procedure LoadFromFile(const FileName: string)`

Visibility: public

Description: `LoadFromFile` creates a file stream with `FileName` as the name of the file to open, en then calls LoadFromStream (473) to read the contents of the blob field from the file. The file is opened in read-only mode.

Errors: If the file does not exist or is nor available for reading, an exception will be raised.

See also: LoadFromStream (473), SaveToFile (473)

16.13.8 TBlobField.LoadFromStream

Synopsis: Load the field's contents from stream.

Declaration: `procedure LoadFromStream(Stream: TStream)`

Visibility: public

Description: `LoadFromStream` can be used to load the contents of the field from a TStream (**??**) descendent. The entire data of the stream will be copied, and the stream will be positioned on the first byte of data, so it must be seekable.

Errors: If the stream is not seekable, an exception will be raised.

See also: SaveToStream (474), LoadFromFile (473)

16.13.9 TBlobField.SaveToFile

Synopsis: Save field contents to a file.

Declaration: `procedure SaveToFile(const FileName: string)`

Visibility: public

Description: `SaveToFile` creates a file stream with `FileName` as the name of the file to open, en then calls SaveToStream (474) to write the contents of the blob field to the file. The file is opened in write mode and is created if it does not yet exist.

Errors: If the file cannot be created or is not available for writing, an exception will be raised.

See also: LoadFromFile (473), SaveToStream (474)

16.13.10 TBlobField.SaveToStream

Synopsis: Save the field's contents to stream.

Declaration: `procedure SaveToStream(Stream: TStream)`

Visibility: public

Description: `SaveToStream` can be used to save the contents of the field to a TStream (**??**) descendent. The entire data of the field will be copied. The stream must of course support writing.

Errors: If the stream is not writable, an exception will be raised.

See also: SaveToFile (473), LoadFromStream (473)

16.13.11 TBlobField.SetFieldType

Synopsis: Set field type.

Declaration: `procedure SetFieldType(AValue: TFieldType);  Override`

Visibility: public

Description: `SetFieldType` is overridden by `TBlobField` to check whether a valid Blob field type is set. If so, it calls the inherited method.

See also: TField.DataType (562)

16.13.12 TBlobField.BlobSize

Synopsis: Size of the current blob.

Declaration: `Property BlobSize : LongInt`

Visibility: public

Access: Read

Description: `BlobSize` is the size (in bytes) of the current contents of the field. It will vary as the dataset's current record moves from record to record.

See also: TField.Size (565), TField.DataSize (562)

16.13.13 TBlobField.Modified

Synopsis: Has the field's contents been modified.

Declaration: `Property Modified : Boolean`

Visibility: public

Access: Read,Write

Description: `Modified` indicates whether the field's contents have been modified for the current record.

See also: TBlobField.LoadFromStream (473)

16.13.14 TBlobField.Value

Synopsis: Return the field's contents as a string.

Declaration: `Property Value : string`

Visibility: public

Access: Read,Write

Description: `Value` is redefined by `TBlobField` as a string value: getting or setting this value will convert the BLOB data to a string, it will return the same value as the TField.AsString (559) property.

See also: TField.Value (566), TField.AsString (559)

16.13.15 TBlobField.Transliterate

Synopsis: Should the contents of the field be transliterated.

Declaration: `Property Transliterate : Boolean`

Visibility: public

Access: Read,Write

Description: `Transliterate` indicates whether the contents of the field should be transliterated (i.e. changed from OEM to non OEM codepage and vice versa) when reading or writing the value. The actual transliteration must be done in the TDataset.Translate (517) method of the dataset to which the field belongs. By default this property is `False`, but it can be set to `True` for BLOB data which contains text in another codepage.

See also: TStringField.Transliterate (642), TDataset.Translate (517)

16.13.16 TBlobField.BlobType

Synopsis: Type of blob.

Declaration: `Property BlobType : TBlobType`

Visibility: published

Access: Read,Write

Description: `BlobType` is an alias for TField.DataType (562), but with a restricted set of values. Setting `BlobType` is equivalent to setting the TField.DataType (562) property.

See also: TField.DataType (562)

16.13.17 TBlobField.Size

Synopsis: Size of the blob field.

Declaration: `Property Size :`

Visibility: published

Access:

Description: `Size` is the size of the blob in the internal memory buffer. It defaults to 0, as the BLOB data is not stored in the internal memory buffer. To get the size of the data in the current record, use the BlobSize (474) property instead.

See also: BlobSize (474)

16.14 TBooleanField

16.14.1 Description

`TBooleanField` is the field class used by TDataset (496) whenever it needs to manage boolean data (TField.DataType (562) equals `ftBoolean`). It overrides some properties and methods of TField (549) to be able to work with boolean data.

It should never be necessary to create an instance of `TBooleanField` manually, a field of this class will be instantiated automatically for each boolean field when a dataset is opened.

See also: TDataset (496), TField (549)

16.14.2 Method overview

Page	Method	Description
476	Create	Create a new instance of the `TBooleanField` class.

16.14.3 Property overview

Page	Properties	Access	Description
477	DisplayValues	rw	Textual representation of the true and false values.
476	Value	rw	Value of the field as a boolean value.

16.14.4 TBooleanField.Create

Synopsis: Create a new instance of the `TBooleanField` class.

Declaration:
```
constructor Create(AOwner: TComponent);  Override
```

Visibility: public

Description: `Create` initializes a new instance of the `TBooleanField` class. It calls the inherited constructor and then sets some TField (549) properties to configure it for working with boolean values.

See also: TField (549)

16.14.5 TBooleanField.Value

Synopsis: Value of the field as a boolean value.

Declaration:
```
Property Value : Boolean
```

Visibility: public

Access: Read,Write

Description: `Value` is redefined from TField.Value (566) by `TBooleanField` as a boolean value. It returns the same value as the TField.AsBoolean (556) property.

See also: TField.AsBoolean (556), TField.Value (566)

16.14.6 TBooleanField.DisplayValues

Synopsis: Textual representation of the true and false values.

Declaration: `Property DisplayValues : string`

Visibility: published

Access: Read,Write

Description: `DisplayValues` contains 2 strings, separated by a semicolon (;) which are used to display the `True` and `False` values of the fields. The first string is used for `True` values, the second value is used for `False` values. If only one value is given, it will serve as the representation of the `True` value, the `False` value will be represented as an empty string.

A value of `Yes;No` will result in `True` values being displayed as 'Yes', and `False` values as 'No'. When writing the value of the field as a string, the string will be compared (case insensitively) with the value for `True`, and if it matches, the field's value will be set to `True`. After this it will be compared to the value for `False`, and if it matches, the field's value will be set to `False`. If the text matches neither of the two values, an exception will be raised.

See also: TField.AsString (559), TField.Text (565)

16.15 TByteField

16.15.1 Description

`TByteField` is instantiated when a dataset must manage a field with 8-bit unsigned data: the data type `ftByte`. It overrides some methods of TField (549) to handle `Byte` data, and sets some of the properties to values for `Byte` data. It also introduces some methods and properties specific to integer data such as MinValue (605) and MaxValue (605).

It should never be necessary to create an instance of `TByteField` manually, a field of this class will be instantiated automatically for each integer field when a dataset is opened.

See also: MinValue (605), MaxValue (605)

16.15.2 Method overview

Page	Method	Description
477	Create	Create new instance of `TByteField`.

16.15.3 TByteField.Create

Synopsis: Create new instance of `TByteField`.

Declaration: `constructor Create(AOwner: TComponent); Override`

Visibility: public

Description: `Create` calls the inherited constructor and sets the values of the MinValue (605)MaxValue (605) and TField.DataType (562) properties.

See also: MinValue (605), MaxValue (605), TField.DataType (562)

16.16 TBytesField

16.16.1 Description

`TBytesField` is the class used when a dataset must manage data of fixed-size binary type. (TField.DataType (562) equals `ftBytes`). It initializes some of the properties of the TField (549) class to be able to work with fixed-size byte fields.

It should never be necessary to create an instance of `TBytesField` manually, a field of this class will be instantiated automatically for each binary data field when a dataset is opened.

See also: TDataset (496), TField (549), TVarBytesField (644)

16.16.2 Method overview

Page	Method	Description
478	Create	Create a new instance of a `TBytesField` class.

16.16.3 TBytesField.Create

Synopsis: Create a new instance of a `TBytesField` class.

Declaration: `constructor Create(AOwner: TComponent);  Override`

Visibility: public

Description: `Create` initializes a new instance of the `TBytesField` class. It calls the inherited destructor, and then sets some TField (549) properties to configure the instance for working with binary data values.

See also: TField (549)

16.17 TCheckConstraint

16.17.1 Description

`TCheckConstraint` can be used to store the definition of a record-level constraint. It does not enforce the constraint, it only stores the constraint's definition. The constraint can come from several sources: an imported constraints from the database, usually stored in the TCheckConstraint.ImportedConstraint (480) property , or a constraint enforced by the user on a particular dataset instance stored in TCheck-Constraint.CustomConstraint (479)

See also: TCheckConstraints (480), TCheckConstraint.ImportedConstraint (480), TCheckConstraint.CustomConstraint (479)

16.17.2 Method overview

Page	Method	Description
479	Assign	Assign one constraint to another.

16.17.3 Property overview

Page	Properties	Access	Description
479	CustomConstraint	rw	User-defined constraint.
479	ErrorMessage	rw	Message to display when the constraint is violated.
480	FromDictionary	rw	True if the constraint is imported from a datadictionary.
480	ImportedConstraint	rw	Constraint imported from the database engine.

16.17.4 TCheckConstraint.Assign

Synopsis: Assign one constraint to another.

Declaration: `procedure Assign(Source: TPersistent);  Override`

Visibility: public

Description: `Assign` is overridden by `TCheckConstraint` to copy all published properties if `Source` is also a `TCheckConstraint` instance.

Errors: If `Source` is not an instance of `TCheckConstraint`, an exception may be thrown.

See also: TCheckConstraint.ImportedConstraint (480), TCheckConstraint.CustomConstraint (479)

16.17.5 TCheckConstraint.CustomConstraint

Synopsis: User-defined constraint.

Declaration: `Property CustomConstraint : string`

Visibility: published

Access: Read,Write

Description: `CustomConstraint` is an SQL expression with an additional user-defined constraint. The expression should be enforced by a TDataset (496) descendent when data is posted to the dataset. If the constraint is violated, then the dataset should raise an exception, with message as specified in TCheckConstraint.ErrorMessage (479)

See also: TCheckConstraint.ErrorMessage (479)

16.17.6 TCheckConstraint.ErrorMessage

Synopsis: Message to display when the constraint is violated.

Declaration: `Property ErrorMessage : string`

Visibility: published

Access: Read,Write

Description: `ErrorMessage` is used as the message when the dataset instance raises an exception if the constraint is violated.

See also: TCheckConstraint.CustomConstraint (479)

16.17.7 TCheckConstraint.FromDictionary

Synopsis: True if the constraint is imported from a datadictionary.

Declaration: `Property FromDictionary : Boolean`

Visibility: published

Access: Read,Write

Description: `FromDictionary` indicates whether a constraint is imported from a data dictionary. This can be set by TDataset (496) descendents to indicate the source of the constraint, but is otherwise ignored.

See also: TCheckConstraint.ImportedConstraint (480)

16.17.8 TCheckConstraint.ImportedConstraint

Synopsis: Constraint imported from the database engine.

Declaration: `Property ImportedConstraint : string`

Visibility: published

Access: Read,Write

Description: `ImportedConstraint` is a constraint imported from the database engine: it will not be enforced locally by the TDataset (496) descendent.

See also: TCheckConstraint.CustomConstraint (479)

16.18 TCheckConstraints

16.18.1 Description

`TCheckConstraints` is a `TCollection` descendent which keeps a collection of TCheckConstraint (478) items. It overrides the Add (481) method to return a `TCheckConstraint` instance.

See also: TCheckConstraint (478)

16.18.2 Method overview

Page	Method	Description
481	Add	Add new `TCheckConstraint` item to the collection.
481	Create	Create a new instance of the `TCheckConstraints` class.

16.18.3 Property overview

Page	Properties	Access	Description
481	Items	rw	Indexed access to the items in the collection.

16.18.4 TCheckConstraints.Create

Synopsis: Create a new instance of the `TCheckConstraints` class.

Declaration: `constructor Create(AOwner: TPersistent)`

Visibility: public

Description: `Create` initializes a new instance of the `TCheckConstraints` class. The `AOwner` argument is usually the TDataset (496) instance for which the data is managed. It is kept for future reference. After storing the owner, the inherited constructor is called with the TCheckConstraint (478) class pointer.

See also: TCheckConstraint (478), TDataset (496)

16.18.5 TCheckConstraints.Add

Synopsis: Add new `TCheckConstraint` item to the collection.

Declaration: `function Add : TCheckConstraint`

Visibility: public

Description: `Add` is overridden by `TCheckConstraint` to add a new TCheckConstraint (478) instance to the collection. it returns the newly added instance.

See also: TCheckConstraint (478), #rtl.classes.TCollection.Add (**??**)

16.18.6 TCheckConstraints.Items

Synopsis: Indexed access to the items in the collection.

Declaration: `Property Items[Index: LongInt]: TCheckConstraint; default`

Visibility: public

Access: Read,Write

Description: `Items` is overridden by `TCheckConstraints` to provide type-safe access to the items in the collection. The `index` is zero-based, so it runs from 0 to `Count-1`.

See also: #rtl.classes.TCollection.Items (**??**)

16.19 TCurrencyField

16.19.1 Description

`TCurrencyField` is the field class used by TDataset (496) when it needs to manage currency-valued data.(TField.Datatype (562) equals `ftCurrency`). It simply sets some Tfield (549) properties to be able to work with currency data.

It should never be necessary to create an instance of `TCurrencyField` manually, a field of this class will be instantiated automatically for each currency field when a dataset is opened.

See also: TField (549), TDataset (496)

16.19.2 Method overview

Page	Method	Description
482	Create	Create a new instance of a `TCurrencyField`.

16.19.3 Property overview

Page	Properties	Access	Description
482	Currency		Is the field a currency field.

16.19.4 TCurrencyField.Create

Synopsis: Create a new instance of a `TCurrencyField`.

Declaration: `constructor Create(AOwner: TComponent); Override`

Visibility: public

Description: `Create` initializes a new instance of `TCurrencyField`. It calls the inherited constructor and then sets some properties (TCurrencyField.Currency (482)) to be able to work with currency data.

See also: TField (549), TCurrencyField.Currency (482)

16.19.5 TCurrencyField.Currency

Synopsis: Is the field a currency field.

Declaration: `Property Currency :`

Visibility: published

Access:

Description: `Currency` is inherited from TFloatField.Currency (591) but is initialized to `True` by the `TCurrencyField` constructor. It can be set to `False` if the contents of the field is of type currency, but does not represent an amount of currency.

See also: TFloatField.Currency (591)

16.20 TCustomConnection

16.20.1 Description

`TCustomConnection` must be used for all database classes that need a connection to a server. The class introduces some methods and classes to activate the connection (Open (483)) and to deactivate the connection (TCustomConnection.Close (483)), plus a property to inspect the state (Connected (484)) of the connected.

See also: TCustomConnection.Open (483), TCustomConnection.Close (483), TCustomConnection.Connected (484)

16.20.2 Method overview

Page	Method	Description
483	Close	Close the connection.
483	Destroy	Remove the `TCustomconnection` instance from memory.
483	Open	Makes the connection to the server.

16.20.3 Property overview

Page	Properties	Access	Description
485	AfterConnect	rw	Event triggered after a connection is made.
485	AfterDisconnect	rw	Event triggered after a connection is closed.
486	BeforeConnect	rw	Event triggered before a connection is made.
486	BeforeDisconnect	rw	Event triggered before a connection is closed.
484	Connected	rw	Is the connection established or not.
484	DataSetCount	r	Number of datasets connected to this connection.
484	DataSets	r	Datasets linked to this connection.
485	LoginPrompt	rw	Should the `OnLogin` be triggered.
486	OnCloseError	rw	
486	OnLogin	rw	Event triggered when a login prompt is shown.

16.20.4 TCustomConnection.Close

Synopsis: Close the connection.

Declaration: `procedure Close(ForceClose: Boolean)`

Visibility: public

Description: `Close` closes the connection with the server if it was connected. Calling this method first triggers the BeforeDisconnect (486) event. If an exception is raised during the execution of that event handler, the disconnect process is aborted. After calling this event, the connection is actually closed. After the connection was closed, the AfterDisconnect (485) event is triggered.

Calling the `Close` method is equivalent to setting the Connected (484) property to `False`.

If `ForceClose` is `True` then the descendent should ignore errors from the underlying connection, allowing all datasets to be closed properly.

Errors: If the connection cannot be broken for some reason, an EDatabaseError (458) exception will be raised.

See also: TCustomConnection.BeforeDisconnect (486), TCustomConnection.AfterDisconnect (485), TCustomConnection.Open (483), TCustomConnection.Connected (484)

16.20.5 TCustomConnection.Destroy

Synopsis: Remove the `TCustomconnection` instance from memory.

Declaration: `destructor Destroy;  Override`

Visibility: public

Description: `Destroy` closes the connection, and then calls the inherited destructor.

Errors: If an exception is raised during the disconnect process, an exception will be raise, and the instance is not removed from memory.

See also: TCustomConnection.Close (483)

16.20.6 TCustomConnection.Open

Synopsis: Makes the connection to the server.

Declaration: `procedure Open`

Visibility: public

Description: `Open` establishes the connection with the server if it was not yet connected. Calling this method first triggers the BeforeConnect (486) event. If an exception is raised during the execution of that event handler, the connect process is aborted. If LoginPrompt (485) is `True`, the OnLogin (486) event handler is called. Only after this event, the connection is actually established. After the connection was established, the AfterConnect (485) event is triggered.

Calling the `Open` method is equivalent to setting the Connected (484) property to `True`.

Errors: If an exception is raised during the `BeforeConnect` or `OnLogin` handlers, the connection is not actually established.

See also: TCustomConnection.BeforeConnect (486), TCustomConnection.LoginPrompt (485), TCustomConnection.OnLogin (486), TCustomConnection.AfterConnect (485), TCustomConnection.Connected (484)

16.20.7 TCustomConnection.DataSetCount

Synopsis: Number of datasets connected to this connection.

Declaration: `Property DataSetCount : LongInt`

Visibility: public

Access: Read

Description: `DatasetCount` is the number of datasets connected to this connection component. The actual datasets are available through the Datasets (484) array property. As implemented in `TCustomConnection`, this property is always zero. Descendent classes implement the actual count.

See also: TDataset (496), TCustomConnection.Datasets (484)

16.20.8 TCustomConnection.DataSets

Synopsis: Datasets linked to this connection.

Declaration: `Property DataSets[Index: LongInt]: TDataSet`

Visibility: public

Access: Read

Description: `Datasets` allows indexed access to the datasets connected to this connection. `Index` is a zero-based indexed, it's maximum value is DatasetCount-1 (484).

See also: DatasetCount (484)

16.20.9 TCustomConnection.Connected

Synopsis: Is the connection established or not.

Declaration: `Property Connected : Boolean`

Visibility: published

Access: Read,Write

Description: `Connected` is `True` if the connection to the server is established, `False` if it is disconnected. The property can be set to `True` to establish a connection (equivalent to calling TCustomConnection.Open (483), or to `False` to break it (equivalent to calling TCustomConnection.Close (483).

See also: TCustomConnection.Open (483), TCustomConnection.Close (483)

16.20.10 TCustomConnection.LoginPrompt

Synopsis: Should the `OnLogin` be triggered.

Declaration: `Property LoginPrompt : Boolean`

Visibility: published

Access: Read,Write

Description: `LoginPrompt` can be set to `True` if the `OnLogin` handler should be called when the `Open` method is called. If it is not `True`, then the event handler is not called.

See also: TCustomConnection.OnLogin (486)

16.20.11 TCustomConnection.AfterConnect

Synopsis: Event triggered after a connection is made.

Declaration: `Property AfterConnect : TNotifyEvent`

Visibility: published

Access: Read,Write

Description: `AfterConnect` is called after a connection is successfully established in TCustomConnection.Open (483). It can be used to open datasets, or indicate a connection status change.

See also: TCustomConnection.Open (483), TCustomConnection.BeforeConnect (486), TCustomConnection.OnLogin (486)

16.20.12 TCustomConnection.AfterDisconnect

Synopsis: Event triggered after a connection is closed.

Declaration: `Property AfterDisconnect : TNotifyEvent`

Visibility: published

Access: Read,Write

Description: `AfterDisConnect` is called after a connection is successfully closed in TCustomConnection.Close (483). It can be used for instance to indicate a connection status change.

See also: TCustomConnection.Close (483), TCustomConnection.BeforeDisconnect (486)

16.20.13 TCustomConnection.BeforeConnect

Synopsis: Event triggered before a connection is made.

Declaration: `Property BeforeConnect : TNotifyEvent`

Visibility: published

Access: Read,Write

Description: `BeforeConnect` is called before a connection is attempted in TCustomConnection.Open (483). It can be used to set connection parameters, or to abort the establishing of the connection: if an exception is raised during this event, the connection attempt is aborted.

See also: TCustomConnection.Open (483), TCustomConnection.AfterConnect (485), TCustomConnection.OnLogin (486)

16.20.14 TCustomConnection.BeforeDisconnect

Synopsis: Event triggered before a connection is closed.

Declaration: `Property BeforeDisconnect : TNotifyEvent`

Visibility: published

Access: Read,Write

Description: `BeforeDisConnect` is called before a connection is closed in TCustomConnection.Close (483). It can be used for instance to check for unsaved changes, to save those changes, or to abort the disconnect operation: if an exception is raised during the event handler, the disconnect operation is aborted entirely.

See also: TCustomConnection.Close (483), TCustomConnection.AfterDisconnect (485)

16.20.15 TCustomConnection.OnLogin

Synopsis: Event triggered when a login prompt is shown.

Declaration: `Property OnLogin : TLoginEvent`

Visibility: published

Access: Read,Write

Description: `OnLogin` is triggered when the connection needs a login prompt during the call: it is triggered when the LoginPrompt (485) property is `True`, after the TCustomConnection.BeforeConnect (486) event, but before the connection is actually established.

See also: TCustomConnection.BeforeConnect (486), TCustomConnection.LoginPrompt (485), TCustomConnection.Open (483)

16.20.16 TCustomConnection.OnCloseError

Declaration: `Property OnCloseError : TCloseErrorEvent`

Visibility: published

Access: Read,Write

16.21 TDatabase

16.21.1 Description

`TDatabase` is a component whose purpose is to provide a connection to an external database engine, not to provide the database itself. This class provides generic methods for attachment to databases and querying their contents; the details of the actual connection are handled by database-specific components (such as SQLDb for SQL-based databases, or DBA for DBASE/FoxPro style databases).

Like TDataset (496), TDatabase is an abstract class. It provides methods to keep track of datasets connected to the database, and to close these datasets when the connection to the database is closed. To this end, it introduces a Connected (490) boolean property, which indicates whether a connection to the database is established or not. The actual logic to establish a connection to a database must be implemented by descendent classes.

See also: TDataset (496), TDatabase (487)

16.21.2 Method overview

Page	Method	Description
488	CloseDataSets	Close all connected datasets.
488	CloseTransactions	End all transactions.
487	Create	Initialize a new `TDatabase` class instance.
488	Destroy	Remove a `TDatabase` instance from memory.
489	EndTransaction	End an active transaction.
488	StartTransaction	Start a new transaction.

16.21.3 Property overview

Page	Properties	Access	Description
490	Connected	rw	Is the database connected.
490	DatabaseName	rw	Database name or path.
489	Directory	rw	Directory for the database.
490	IsSQLBased	r	Is the database SQL based.
490	KeepConnection	rw	Should the connection be kept active.
491	Params	rw	Connection parameters.
489	TransactionCount	r	Number of transaction components connected to this database.
489	Transactions	r	Indexed access to all transaction components connected to this database.

16.21.4 TDatabase.Create

Synopsis: Initialize a new `TDatabase` class instance.

Declaration:
```
constructor Create(AOwner: TComponent);  Override
```

Visibility: public

Description: `Create` initializes a new instance of the `TDatabase` class. It allocates some resources and then calls the inherited constructor.

See also: TDBDataset (542), TDBTransaction (543), TDatabase.Destroy (488)

16.21.5 TDatabase.Destroy

Synopsis: Remove a `TDatabase` instance from memory.

Declaration: `destructor Destroy; Override`

Visibility: public

Description: `Destroy` starts by disconnecting the database (thus closing all datasets and ending all transactions), then notifies all connected datasets and transactions that it is about to be released. After this, it releases all resources used by the `TDatabase` instance

See also: TDatabase.CloseDatasets (488)

16.21.6 TDatabase.CloseDataSets

Synopsis: Close all connected datasets.

Declaration: `procedure CloseDataSets`

Visibility: public

Description: `CloseDatasets` closes all connected datasets. It is called automatically when the connection is closed.

See also: TCustomConnection.Close (483), TDatabase.CloseTransactions (488)

16.21.7 TDatabase.CloseTransactions

Synopsis: End all transactions.

Declaration: `procedure CloseTransactions`

Visibility: public

Description: `CloseTransaction` calls TDBTransaction.EndTransaction (543) on all connected transactions. It is called automatically when the connection is closed, after all datasets are closed.

See also: TCustomConnection.Close (483), TDatabase.CloseDatasets (488)

16.21.8 TDatabase.StartTransaction

Synopsis: Start a new transaction.

Declaration: `procedure StartTransaction; Virtual; Abstract`

Visibility: public

Description: `StartTransaction` must be implemented by descendent classes to start a new transaction. This method is provided for Delphi compatibility: new applications should use a TDBTransaction (543) component instead and invoke the TDBTransaction.StartTransaction (543) method.

See also: TDBTransaction (543), TDBTransaction.StartTransaction (543)

16.21.9 TDatabase.EndTransaction

Synopsis: End an active transaction.

Declaration: `procedure EndTransaction; Virtual; Abstract`

Visibility: public

Description: `EndTransaction` must be implemented by descendent classes to end an active transaction. This method is provided for Delphi compatibility: new applications should use a TDBTransaction (543) component instead and invoke the TDBTransaction.EndTransaction (543) method.

See also: TDBTransaction (543), TDBTransaction.EndTransaction (543)

16.21.10 TDatabase.TransactionCount

Synopsis: Number of transaction components connected to this database.

Declaration: `Property TransactionCount : LongInt`

Visibility: public

Access: Read

Description: `TransactionCount` is the number of transaction components which are connected to this database instance. It is the upper bound for the TDatabase.Transactions (489) array property.

See also: TDatabase.Transactions (489)

16.21.11 TDatabase.Transactions

Synopsis: Indexed access to all transaction components connected to this database.

Declaration: `Property Transactions[Index: LongInt]: TDBTransaction`

Visibility: public

Access: Read

Description: `Transactions` provides indexed access to the transaction components connected to this database. The `Index` is zero based: it runs from 0 to `TransactionCount-1`.

See also: TDatabase.TransactionCount (489)

16.21.12 TDatabase.Directory

Synopsis: Directory for the database.

Declaration: `Property Directory : string`

Visibility: public

Access: Read,Write

Description: `Directory` is provided for Delphi compatibility: it indicates (for Paradox and dBase based databases) the directory where the database files are located. It is not used in the Free Pascal implementation of TDatabase (487).

See also: TDatabase.Params (491), TDatabase.IsSQLBased (490)

16.21.13 TDatabase.IsSQLBased

Synopsis: Is the database SQL based.

Declaration: `Property IsSQLBased : Boolean`

Visibility: public

Access: Read

Description: `IsSQLbased` is a read-only property which indicates whether a property is SQL-Based, i.e. whether the database engine accepts SQL commands.

See also: TDatabase.Params (491), TDatabase.Directory (489)

16.21.14 TDatabase.Connected

Synopsis: Is the database connected.

Declaration: `Property Connected : Boolean`

Visibility: published

Access: Read,Write

Description: `Connected` is simply promoted to published property from TCustomConnection.Connected (484).

See also: TCustomConnection.Connected (484)

16.21.15 TDatabase.DatabaseName

Synopsis: Database name or path.

Declaration: `Property DatabaseName : string`

Visibility: published

Access: Read,Write

Description: `DatabaseName` specifies the path of the database. For directory-based databases this will be the same as the Directory (489) property. For other databases this will be the name of a known pre-configured connection, or the location of the database file.

See also: TDatabase.Directory (489), TDatabase.Params (491)

16.21.16 TDatabase.KeepConnection

Synopsis: Should the connection be kept active.

Declaration: `Property KeepConnection : Boolean`

Visibility: published

Access: Read,Write

Description: `KeepConnection` is provided for Delphi compatibility, and is not used in the Free Pascal implementation of `TDatabase`.

See also: TDatabase.Params (491)

16.21.17 TDatabase.Params

Synopsis: Connection parameters.

Declaration: `Property Params : TStrings`

Visibility: published

Access: Read,Write

Description: `Params` is a catch-all storage mechanism for database connection parameters. It is a list of strings in the form of `Name=Value` pairs. Which name/value pairs are supported depends on the `TDatabase` descendent, but the `user_name` and `password` parameters are commonly used to store the login credentials for the database.

See also: TDatabase.Directory (489), TDatabase.DatabaseName (490)

16.22 TDataLink

16.22.1 Description

`TDataLink` is used by GUI controls or datasets in a master-detail relationship to handle data events coming from a TDatasource (536) instance. It is a class that exists for component programmers, application coders should never need to use `TDataLink` or one of it's descendents.

DB-Aware Component coders must use a `TDatalink` instance to handle all communication with a TDataset (496) instance, rather than communicating directly with the dataset. `TDataLink` contains methods which are called by the various events triggered by the dataset. Inversely, it has some methods to trigger actions in the dataset.

`TDatalink` is an abstract class; it is never used directly. Instead, a descendent class is used which overrides the various methods that are called in response to the events triggered by the dataset. Examples are .

See also: TDataset (496), TDatasource (536), TDetailDatalink (547), TMasterDataLink (609)

16.22.2 Method overview

Page	Method	Description
492	Create	Initialize a new instance of `TDataLink`.
492	Destroy	Remove an instance of `TDatalink` from memory.
492	Edit	Set the dataset in edit mode, if possible.
493	ExecuteAction	Execute action.
493	UpdateAction	Update handler for actions.
493	UpdateRecord	Called when the data in the dataset must be updated.

16.22.3 Property overview

Page	Properties	Access	Description
493	Active	r	Is the link active.
494	ActiveRecord	rw	Currently active record.
494	BOF	r	Is the dataset at the first record.
494	BufferCount	rw	Set to the number of record buffers this datalink needs.
495	DataSet	r	Dataset this datalink is connected to.
495	DataSource	rw	Datasource this datalink is connected to.
495	DataSourceFixed	rw	Can the datasource be changed.
495	Editing	r	Is the dataset in edit mode.
496	Eof	r	
496	ReadOnly	rw	Is the link readonly.
496	RecordCount	r	Number of records in the buffer of the dataset.

16.22.4 TDataLink.Create

Synopsis: Initialize a new instance of `TDataLink`.

Declaration: `constructor Create`

Visibility: public

Description: `Create` calls the inherited constructor and then initializes some fields. In particular, it sets the buffercount to 1.

See also: TDatalink.Destroy (492)

16.22.5 TDataLink.Destroy

Synopsis: Remove an instance of `TDatalink` from memory.

Declaration: `destructor Destroy;  Override`

Visibility: public

Description: `Destroy` cleans up the `TDatalink` instance (in particular, it removes itself from the datasource it is coupled to), and then calls the inherited destructor.

See also: TDatalink.Destroy (492)

16.22.6 TDataLink.Edit

Synopsis: Set the dataset in edit mode, if possible.

Declaration: `function Edit : Boolean`

Visibility: public

Description: `Edit` attempts to put the dataset in edit mode. It returns `True` if this operation succeeded, `False` if not. To this end, it calls the Edit (537) method of the DataSource (495) to which the datalink instance is coupled. If the TDatasource.AutoEdit (538) property is `False` then this operation will not succeed, unless the dataset is already in edit mode. GUI controls should always respect the result of this function, and not allow the user to edit data if this function returned `false`.

See also: TDatasource (536), TDatalink.Datasource (495), TDatasource.Edit (537), TDatasource.AutoEdit (538)

16.22.7 TDataLink.UpdateRecord

Synopsis: Called when the data in the dataset must be updated.

Declaration: `procedure UpdateRecord`

Visibility: public

Description: `Updaterecord` is called when the dataset expects the GUI controls to post any pending changes to the dataset. This method guards against recursive behaviour: while an `UpdateRecord` is in progress, the TDatalink.RecordChange (491) notification (which could result from writing data to the dataset) will be blocked.

See also: TDatalink.RecordChange (491)

16.22.8 TDataLink.ExecuteAction

Synopsis: Execute action.

Declaration: `function ExecuteAction(Action: TBasicAction) : Boolean;  Virtual`

Visibility: public

Description: `ExecuteAction` implements action support. It should never be necessary to call `ExecuteAction` from program code, as it is called automatically whenever a target control needs to handle an action. This method must be overridden in case any additional action must be taken when the action must be executed. The implementation in `TDatalink` checks if the action handles the datasource, and then calls `Action.ExecuteTarget`, passing it the datasource. If so, it returns `True`.

See also: TDatalink.UpdateAction (493)

16.22.9 TDataLink.UpdateAction

Synopsis: Update handler for actions.

Declaration: `function UpdateAction(Action: TBasicAction) : Boolean;  Virtual`

Visibility: public

Description: `UpdateAction` implements action update support. It should never be necessary to call `UpdateAction` from program code, as it is called automatically whenever a target control needs to update an action. This method must be overridden in case any specific action must be taken when the action must be updated. The implementation in `TDatalink` checks if the action handles the datasource, and then calls `Action.UpdateTarget`, passing it the datasource. If so, it returns `True`.

See also: TDataLink.ExecuteAction (493)

16.22.10 TDataLink.Active

Synopsis: Is the link active.

Declaration: `Property Active : Boolean`

Visibility: public

Access: Read

Description: `Active` determines whether the events of the dataset are passed on to the control connected to the actionlink. If it is set to `False`, then no events are passed between control and dataset. It is set to TDataset.Active (526) whenever the DataSource (495) property is set.

See also: TDatalink.Datasource (495), TDatalink.ReadOnly (496), TDataset.Active (526)

16.22.11 TDataLink.ActiveRecord

Synopsis: Currently active record.

Declaration: `Property ActiveRecord : Integer`

Visibility: public

Access: Read,Write

Description: `ActiveRecord` returns the index of the active record in the dataset's record buffer for this datalink.

See also: TDatalink.BOF (494), TDatalink.EOF (496)

16.22.12 TDataLink.BOF

Synopsis: Is the dataset at the first record.

Declaration: `Property BOF : Boolean`

Visibility: public

Access: Read

Description: `BOF` returns TDataset.BOF (518) if the dataset is available, `True` otherwise.

See also: TDatalink.EOF (496), TDataset.BOF (518)

16.22.13 TDataLink.BufferCount

Synopsis: Set to the number of record buffers this datalink needs.

Declaration: `Property BufferCount : Integer`

Visibility: public

Access: Read,Write

Description: `BufferCount` can be set to the number of buffers that the dataset should manage on behalf of the control connected to this datalink. By default, this is 1. Controls that must display more than 1 buffer (such as grids) can set this to a higher value.

See also: TDataset.ActiveBuffer (502), TDatalink.ActiveRecord (494)

16.22.14 TDataLink.DataSet

Synopsis: Dataset this datalink is connected to.

Declaration: `Property DataSet : TDataSet`

Visibility: public

Access: Read

Description: `Dataset` equals `Datasource.Dataset` if the datasource is set, or `Nil` otherwise.

See also: TDatalink.DataSource (495), TDataset (496)

16.22.15 TDataLink.DataSource

Synopsis: Datasource this datalink is connected to.

Declaration: `Property DataSource : TDataSource`

Visibility: public

Access: Read,Write

Description: `Datasource` should be set to a TDatasource (536) instance to get access to the dataset it is connected to. A datalink never points directly to a TDataset (496) instance, always to a datasource. When the datasource is enabled or disabled, all `TDatalink` instances connected to it are enabled or disabled at once.

See also: TDataset (496), TDatasource (536)

16.22.16 TDataLink.DataSourceFixed

Synopsis: Can the datasource be changed.

Declaration: `Property DataSourceFixed : Boolean`

Visibility: public

Access: Read,Write

Description: `DatasourceFixed` can be set to `True` to prevent changing of the DataSource (495) property. When lengthy operations are in progress, this can be done to prevent user code (e.g. event handlers) from changing the datasource property which might interfere with the operation in progress.

See also: TDataLink.DataSource (495)

16.22.17 TDataLink.Editing

Synopsis: Is the dataset in edit mode.

Declaration: `Property Editing : Boolean`

Visibility: public

Access: Read

Description: `Editing` determines whether the dataset is in one of the edit states (dsEdit,dsInsert). It can be set into this mode by calling the TDatalink.Edit (492) method. Never attempt to set the dataset in editing mode directly. The `Edit` method will perform the needed checks prior to setting the dataset in edit mode and will return `True` if the dataset was successfully set in the editing state.

See also: TDatalink.Edit (492), TDataset.Edit (507)

16.22.18 TDataLink.Eof

Synopsis:

Declaration: `Property Eof : Boolean`

Visibility: public

Access: Read

Description: `EOF` returns TDataset.EOF (520) if the dataset is available, `True` otherwise.

See also: TDatalink.BOF (494), TDataset.EOF (520)

16.22.19 TDataLink.ReadOnly

Synopsis: Is the link readonly.

Declaration: `Property ReadOnly : Boolean`

Visibility: public

Access: Read,Write

Description: `ReadOnly` can be set to `True` to indicate that the link is read-only, i.e. the connected control will not modify the dataset. Methods as TDatalink.Edit (492) will check this property and fail if the link is read-only. This setting has no effect on the communication of dataset events to the datalink: the TDatalink.Active (493) property can be used to disable delivery of events to the datalink.

See also: TDatalink.Active (493), TDatalink.edit (492)

16.22.20 TDataLink.RecordCount

Synopsis: Number of records in the buffer of the dataset.

Declaration: `Property RecordCount : Integer`

Visibility: public

Access: Read

Description: `RecordCount` returns the number of records in the dataset's buffer. It is limited by the TDatalink.BufferCount (494) property: `RecordCount` i s always less than `Buffercount`.

See also: TDatalink.BufferCount (494)

16.23 TDataSet

16.23.1 Description

`TDataset` is the main class of the db unit. This abstract class provides all basic functionality to access data stored in tabular format: The data consists of records, and the data in each record is organised in several fields.

`TDataset` has a buffer to cache a few records in memory, this buffer is used by `TDatasource` to create the ability to use data-aware components.

`TDataset` is an abstract class, which provides the basic functionality to access, navigate through the data and - in case read-write access is available, edit existing or add new records.

TDataset is an abstract class: it does not have the knowledge to store or load the records from whatever medium the records are stored on. Descendants add the functionality to load and save the data. Therefor TDataset is never used directly, one always instantiates a descendent class.

Initially, no data is available: the dataset is inactive. The Open (515) method must be used to fetch data into memory. After this command, the data is available in memory for browsing or editing purposes: The dataset is active (indicated by the TDataset.Active (526) property). Likewise, the Close (505) method can be used to remove the data from memory. Any changes not yet saved to the underlying medium will be lost.

Data is expected to be in tabular format, where each row represents a record. The dataset has an idea of a cursor: this is the current position of the data cursor in the set of rows. Only the data of the current record is available for display or editing purposes. Through the Next (514), Prev (496), First (510) and Last (513) methods, it is possible to navigate through the records. The EOF (520) property will be True if the last row has been reached. Likewise, the BOF (518) property will return True if the first record in the dataset has been reached when navigating backwards. If both properties are empty, then there is no data available. For dataset descendents that support counting the number of records, the RecordCount (523) will be zero.

The Append (503) and Insert (512) methods can be used to insert new records to the set of records. The TDataset.Delete (506) statement is used to delete the current record, and the Edit (507) command must be used to set the dataset in editing mode: the contents of the current record can then be changed. Any changes made to the current record (be it a new or existing record) must be saved by the Post (515) method, or can be undone using the Cancel (504) method.

The data in the various fields properties is available through the Fields (524) array property, giving indexed access to all the fields in a record. The contents of a field is always readable. If the dataset is in one of the editing modes, then the fields can also be written to.

See also: TField (549)

16.23.2 Method overview

Page	Method	Description
502	ActiveBuffer	Currently active memory buffer.
503	Append	Append a new record to the data.
503	AppendRecord	Append a new record to the dataset and fill with data.
504	BookmarkValid	Test whether `ABookMark` is a valid bookmark.
504	Cancel	Cancel the current editing operation.
504	CheckBrowseMode	Check whether the dataset is in browse mode.
504	ClearFields	Clear the values of all fields.
505	Close	Close the dataset.
505	CompareBookmarks	Compare two bookmarks.
505	ControlsDisabled	Check whether the controls are disabled.
502	Create	Create a new `TDataset` instance.
506	CreateBlobStream	Create blob stream.
506	CursorPosChanged	Indicate a change in cursor position.
506	DataConvert	Convert data from/to native format.
506	Delete	Delete the current record.
502	Destroy	Free a `TDataset` instance.
507	DisableControls	Disable event propagation of controls.
507	Edit	Set the dataset in editing mode.
508	EnableControls	Enable event propagation of controls.
508	FieldByName	Search a field by name.
508	FindField	Find a field by name.
509	FindFirst	Find the first active record (deprecated).
509	FindLast	Find the last active record (deprecated).
509	FindNext	Find the next active record (deprecated).
509	FindPrior	Find the previous active record (deprecated).
510	First	Position the dataset on the first record.
510	FreeBookmark	Free a bookmark obtained with `GetBookmark` (deprecated).
510	GetBookmark	Get a bookmark pointer (deprecated).
511	GetCurrentRecord	Copy the data for the current record in a memory buffer.
502	GetFieldData	Get the data for a field.
511	GetFieldList	Return field instances in a list.
511	GetFieldNames	Return a list of all available field names.
511	GotoBookmark	Jump to bookmark.
512	Insert	Insert a new record at the current position.
512	InsertRecord	Insert a new record with given values.
512	IsEmpty	Check if the dataset contains no data.
512	IsLinkedTo	Check whether a datasource is linked to the dataset.
513	IsSequenced	Is the data sequenced.
513	Last	Navigate forward to the last record.
513	Locate	Locate a record based on some key values.
514	Lookup	Search for a record and return matching values.
514	MoveBy	Move the cursor position.
514	Next	Go to the next record in the dataset.
515	Open	Activate the dataset: Fetch data into memory.
515	Post	Post pending edits to the database.
516	Prior	Go to the previous record.
516	Refresh	Refresh the records in the dataset.
516	Resync	Resynchronize the data buffer.
503	SetFieldData	Store the data for a field.
516	SetFields	Set a number of field values at once.
517	Translate	Transliterate a buffer.
517	UpdateCursorPos	Update cursor position.
517	UpdateRecord	Indicate that the record contents have changed.
518	UpdateStatus	Get the update status for the current record.

16.23.3 Property overview

Page	Properties	Access	Description
526	Active	rw	Is the dataset open or closed.
530	AfterCancel	rw	Event triggered after a Cancel operation.
527	AfterClose	rw	Event triggered after the dataset is closed.
531	AfterDelete	rw	Event triggered after a successful Delete operation.
529	AfterEdit	rw	Event triggered after the dataset is put in edit mode.
528	AfterInsert	rw	Event triggered after the dataset is put in insert mode.
527	AfterOpen	rw	Event triggered after the dataset is opened.
529	AfterPost	rw	Event called after changes have been posted to the underlying database.
532	AfterRefresh	rw	Event triggered after the data has been refreshed.
531	AfterScroll	rw	Event triggered after the cursor has changed position.
526	AutoCalcFields	rw	How often should the value of calculated fields be calculated.
530	BeforeCancel	rw	Event triggered before a Cancel operation.
527	BeforeClose	rw	Event triggered before the dataset is closed.
530	BeforeDelete	rw	Event triggered before a Delete operation.
528	BeforeEdit	rw	Event triggered before the dataset is put in edit mode.
528	BeforeInsert	rw	Event triggered before the dataset is put in insert mode.
527	BeforeOpen	rw	Event triggered before the dataset is opened.
529	BeforePost	rw	Event called before changes are posted to the underlying database.
532	BeforeRefresh	rw	Event triggered before the data is refreshed.
531	BeforeScroll	rw	Event triggered before the cursor changes position.
518	BlockReadSize	rw	Number of records to read.
518	BOF	r	Is the cursor at the beginning of the data (on the first record).
518	Bookmark	rw	Get or set the current cursor position.
519	CanModify	r	Can the data in the dataset be modified.
520	DataSource	r	Datasource this dataset is connected to.
520	DefaultFields	r	Is the dataset using persistent fields or not.
520	EOF	r	Indicates whether the last record has been reached.
521	FieldCount	r	Number of fields.
521	FieldDefs	rw	Definitions of available fields in the underlying database.
524	Fields	r	Indexed access to the fields of the dataset.
525	FieldValues	rw	Access to field values based on the field names.
525	Filter	rw	Filter to apply to the data in memory.
525	Filtered	rw	Is the filter active or not.
526	FilterOptions	rw	Options to apply when filtering.
522	Found	r	Check success of one of the `Find` methods.
522	IsUniDirectional	r	Is the dataset unidirectional (i.e. forward scrolling only).
522	Modified	r	Was the current record modified ?
532	OnCalcFields	rw	Event triggered when values for calculated fields must be computed.
533	OnDeleteError	rw	Event triggered when a delete operation fails.
533	OnEditError	rw	Event triggered when an edit operation fails.
534	OnFilterRecord	rw	Event triggered to filter records.
534	OnNewRecord	rw	Event triggered when a new record is created.
534	OnPostError	rw	Event triggered when a post operation fails.
523	RecNo	rw	Current record number.
523	RecordCount	r	Number of records in the dataset.
523	RecordSize	r	Size of the record in memory.
524	SparseArrays	rw	
524	State	r	Current operational state of the dataset.

16.23.4 TDataSet.Create

Synopsis: Create a new `TDataset` instance.

Declaration:
```
constructor Create(AOwner: TComponent);  Override
```

Visibility: public

Description: `Create` initializes a new TDataset (496) instance. It calls the inherited constructor, and then initializes the internal structures needed to manage the dataset (fielddefs, fieldlist, constraints etc.).

See also: TDataset.Destroy (502)

16.23.5 TDataSet.Destroy

Synopsis: Free a `TDataset` instance.

Declaration:
```
destructor Destroy;  Override
```

Visibility: public

Description: `Destroy` removes a `TDataset` instance from memory. It closes the dataset if it was open, clears all internal structures and then calls he inherited destructor.

Errors: An exception may occur during the close operation, in that case, the dataset will not be removed from memory.

See also: TDataset.Close (505), TDataset.Create (502)

16.23.6 TDataSet.ActiveBuffer

Synopsis: Currently active memory buffer.

Declaration:
```
function ActiveBuffer : TRecordBuffer
```

Visibility: public

Description: `ActiveBuffer` points to the currently active memory buffer. It should not be used in application code.

16.23.7 TDataSet.GetFieldData

Synopsis: Get the data for a field.

Declaration:
```
function GetFieldData(Field: TField; Buffer: Pointer) : Boolean
                      ;  Virtual;  Overload
function GetFieldData(Field: TField; Buffer: Pointer;
                      NativeFormat: Boolean) : Boolean;  Virtual
                      ;  Overload
```

Visibility: public

Description: `GetFieldData` should copy the data for field `Field` from the internal dataset memory buffer into the memory pointed to by `Buffer`. This function is not intended for use by end-user applications, and should be used only in descendent classes, where it can be overridden. The function should return `True` if data was available and has been copied, or `False` if no data was available (in which case the field has value `Null`). The `NativeFormat` determines whether the data should be in native format (e.g. whether the date/time values should be in `TDateTime` format).

Errors: No checks are performed on the validity of the memory buffer

See also: TField.DisplayText (563)

16.23.8 TDataSet.SetFieldData

Synopsis: Store the data for a field.

Declaration:
```
procedure SetFieldData(Field: TField; Buffer: Pointer);  Virtual
                       ;  Overload
procedure SetFieldData(Field: TField; Buffer: Pointer;
                       NativeFormat: Boolean);  Virtual;  Overload
```

Visibility: public

Description: `SetFieldData` should copy the data from field `Field`, stored in the memory pointed to by `Buffer` to the dataset memory buffer for the current record. This function is not intended for use by end-user applications, and should be used only in descendent classes, where it can be overridden. The `NativeFormat` determines whether the data is in native format (e.g. whether the date/time values are in `TDateTime` format).

See also: TField.DisplayText (563)

16.23.9 TDataSet.Append

Synopsis: Append a new record to the data.

Declaration:
```
procedure Append
```

Visibility: public

Description: `Append` appends a new record at the end of the dataset. It is functionally equal to the TDataset.Insert (512) call, but the cursor is positioned at the end of the dataset prior to performing the insert operation. The same events occur as when the `Insert` call is made.

See also: TDataset.Insert (512), TDataset.Edit (507)

16.23.10 TDataSet.AppendRecord

Synopsis: Append a new record to the dataset and fill with data.

Declaration:
```
procedure AppendRecord(const Values: Array of const)
```

Visibility: public

Description: `AppendRecord` first calls `Append` to add a new record to the dataset. It then copies the values in `Values` to the various fields (using TDataset.SetFields (516)) and attempts to post the record using TDataset.Post (515). If all went well, the result is that the values in `Values` have been added as a new record to the dataset.

Errors: Various errors may occur (not supplying a value for all required fields, invalid values) and may cause an exception. This may leave the dataset in editing mode.

See also: TDataset.Append (503), TDataset.SetFields (516), TDataset.Post (515)

16.23.11 TDataSet.BookmarkValid

Synopsis: Test whether `ABookMark` is a valid bookmark.

Declaration: `function BookmarkValid(ABookmark: TBookMark) : Boolean;  Virtual`

Visibility: public

Description: `BookmarkValid` returns `True` if `ABookMark` is a valid bookmark for the dataset. Various operations can render a bookmark invalid: changing the sort order, closing and re-opening the dataset. `BookmarkValid` always returns `False` in `TDataset`. Descendent classes must override this method to do an actual test.

Errors: If the bookmark is a completely arbitrary pointer, an exception may be raised.

See also: TDataset.GetBookmark (510), TDataset.SetBookmark (496), TDataset.FreeBookmark (510), TDataset.BookmarkAvailable (496)

16.23.12 TDataSet.Cancel

Synopsis: Cancel the current editing operation.

Declaration: `procedure Cancel;  Virtual`

Visibility: public

Description: `Cancel` cancels the current editing operation and sets the dataset again in browse mode. This operation triggers the TDataset.BeforeCancel (530) and TDataset.AfterCancel (530) events. If the dataset was sin insert mode, then the TDataset.BeforeScroll (531) and TDataset.AfterScroll (531) events are triggered after and respectively before the `BeforeCancel` and `AfterCancel` events.

If the dataset was not in one of the editing modes when `Cancel` is called, then nothing will happen.

See also: TDataset.State (524), TDataset.Append (503), TDataset.Insert (512), TDataset.Edit (507)

16.23.13 TDataSet.CheckBrowseMode

Synopsis: Check whether the dataset is in browse mode.

Declaration: `procedure CheckBrowseMode`

Visibility: public

Description: `CheckBrowseMode` will force the dataset to browse mode (`State=dsBrowse`) if it is active. If it is not active, an EDatabaseError (458) exception is raised. If it is active, but in an edit state, then TDataset.UpdateRecord (517) is called, and if the TDataset.Modified (522) property is true, a TDataset.Post (515) is performed, else TDataset.Cancel (504) is called.

See also: TDataset.State (524), TDataset.Post (515), TDataset.Cancel (504), TDataset.UpdateRecord (517), TDataset.Modified (522)

16.23.14 TDataSet.ClearFields

Synopsis: Clear the values of all fields.

Declaration: `procedure ClearFields`

Visibility: public

Description: `ClearFields` clears the values of all fields.

Errors: If the dataset is not in editing mode (`State in dsEditmodes`), then an EDatabaseError (458) exception will be raised.

See also: TDataset.State (524), TField.Clear (553)

16.23.15 TDataSet.Close

Synopsis: Close the dataset.

Declaration: `procedure Close`

Visibility: public

Description: `Close` closes the dataset if it is open (`Active=True`). This action triggers the TDataset.BeforeClose (527) and TDataset.AfterClose (527) events. If the dataset is not active, nothing happens.

Errors: If an exception occurs during the closing of the dataset, the `AfterClose` event will not be triggered.

See also: TDataset.Active (526), TDataset.Open (515)

16.23.16 TDataSet.ControlsDisabled

Synopsis: Check whether the controls are disabled.

Declaration: `function ControlsDisabled : Boolean`

Visibility: public

Description: `ControlsDisabled` returns `True` if the controls are disabled, i.e. no events are propagated to the controls connected to this dataset. The TDataset.DisableControls (507) call can be used to disable sending of data events to the controls. The sending can be re-enabled with TDataset.EnableControls (508). This mechanism has a counting mechanism: in order to enable sending of events to the controls, `EnableControls` must be called as much as `DisableControls` was called. The `ControlsDisabled` function will return true as long as the internal counter is not zero.

See also: TDataset.DisableControls (507), TDataset.EnableControls (508)

16.23.17 TDataSet.CompareBookmarks

Synopsis: Compare two bookmarks.

Declaration:
```
function CompareBookmarks(Bookmark1: TBookMark; Bookmark2: TBookMark)
                                 : LongInt;  Virtual
```

Visibility: public

Description: `CompareBookmarks` can be used to compare the relative positions of 2 bookmarks. It returns a negative value if `Bookmark1` is located before `Bookmark2`, zero if they refer to the same record, and a positive value if the second bookmark appears before the first bookmark. This function must be overridden by descendent classes of TDataset. The implementation in TDataset always returns zero.

Errors: No checks are performed on the validity of the bookmarks.

See also: TDataset.BookmarkValid (504), TDataset.GetBookmark (510), TDataset.SetBookmark (496)

16.23.18 TDataSet.CreateBlobStream

Synopsis: Create blob stream.

Declaration:
```
function CreateBlobStream(Field: TField; Mode: TBlobStreamMode)
                                  : TStream;  Virtual
```

Visibility: public

Description: `CreateBlobStream` is not intended for use by application programmers. It creates a stream object which can be used to read or write data from a blob field. Instead, application programmers should use the TBlobField.LoadFromStream (473) and TBlobField.SaveToStream (474) methods when reading and writing data from/to BLOB fields. Which operation must be performed on the stream is indicated in the `Mode` parameter, and the `Field` parameter contains the field whose data should be read. The caller is responsible for freeing the stream created by this function.

See also: TBlobField.LoadFromStream (473), TBlobField.SaveToStream (474)

16.23.19 TDataSet.CursorPosChanged

Synopsis: Indicate a change in cursor position.

Declaration:
```
procedure CursorPosChanged
```

Visibility: public

Description: `CursorPosChanged` is not intended for internal use only, and serves to indicate that the current cursor position has changed. (it clears the internal cursor position).

16.23.20 TDataSet.DataConvert

Synopsis: Convert data from/to native format.

Declaration:
```
procedure DataConvert(aField: TField; aSource: Pointer; aDest: Pointer;
                            aToNative: Boolean);  Virtual
```

Visibility: public

Description: `DataConvert` converts the data from field `AField` in buffer `ASource` to native format and puts the result in `ADest`. If the `aToNative` parameter equals `False`, then the data is converted from native format to non-native format. Currently, only date/time/datetime and BCD fields are converted from/to native data. This means the routine handles conversion between `TDateTime` (the native format) and `TDateTimeRec`, and between TBCD and currency (the native format) for BCD fields.

`DataConvert` is used internally by `TDataset` and descendent classes. There should be no need to use this routine in application code.

Errors: No checking on the validity of the buffer pointers is performed. If an invalid pointer is passed, an exception may be raised.

See also: TDataset.GetFieldData (502), TDataset.SetFieldData (503)

16.23.21 TDataSet.Delete

Synopsis: Delete the current record.

Declaration:
```
procedure Delete;  Virtual
```

Visibility: public

Description: `Delete` will delete the current record. This action will trigger the TDataset.BeforeDelete (530), TDataset.BeforeScroll (531), TDataset.AfterDelete (531) and TDataset.AfterScroll (531) events. If the dataset was in edit mode, the edits will be canceled before the delete operation starts.

Errors: If the dataset is empty or read-only, then an EDatabaseError (458) exception will be raised.

See also: TDataset.Cancel (504), TDataset.BeforeDelete (530), TDataset.BeforeScroll (531), TDataset.AfterDelete (531), TDataset.AfterScroll (531)

16.23.22 TDataSet.DisableControls

Synopsis: Disable event propagation of controls.

Declaration: `procedure DisableControls`

Visibility: public

Description: `DisableControls` tells the dataset to stop sending data-related events to the controls. This can be used before starting operations that will cause the current record to change a lot, or before any other lengthy operation that may cause a lot of events to be sent to the controls that show data from the dataset: each event will cause the control to update itself, which is a time-consuming operation that may also cause a lot of flicker on the screen.

The sending of events to the controls can be re-enabled with Tdataset.EnableControls (508). Note that for each call to `DisableControls`, a matching call to `EnableControls` must be made: an internal count is kept and only when the count reaches zero, the controls are again notified of changes to the dataset. It is therefore essential that the call to `EnableControls` is put in a `Finally` block:

```
MyDataset.DisableControls;
Try
  // Do some intensive stuff
Finally
  MyDataset.EnableControls
end;
```

Errors: Failure to call enablecontrols will prevent the controls from receiving updates. The state can be checked with TDataset.ControlsDisabled (505).

See also: TDataset.EnableControls (508), TDataset.ControlsDisabled (505)

16.23.23 TDataSet.Edit

Synopsis: Set the dataset in editing mode.

Declaration: `procedure Edit`

Visibility: public

Description: `Edit` will set the dataset in edit mode: the contents of the current record can then be changed. This action will call the TDataset.BeforeEdit (528) and TDataset.AfterEdit (529) events. If the dataset was already in insert or edit mode, nothing will happen (the events will also not be triggered). If the dataset is empty, this action will execute TDataset.Append (503) instead.

Errors: If the dataset is read-only or not opened, then an EDatabaseError (458) exception will be raised.

See also: TDataset.State (524), TDataset.EOF (520), TDataset.BOF (518), TDataset.Append (503), TDataset.BeforeEdit (528), TDataset.AfterEdit (529)

16.23.24 TDataSet.EnableControls

Synopsis: Enable event propagation of controls.

Declaration: `procedure EnableControls`

Visibility: public

Description: `EnableControls` tells the dataset to resume sending data-related events to the controls. This must be used after a call to TDataset.DisableControls (507) to re-enable updating of controls.

Note that for each call to `DisableControls`, a matching call to `EnableControls` must be made: an internal count is kept and only when the count reaches zero, the controls are again notified of changes to the dataset. It is therefore essential that the call to `EnableControls` is put in a `Finally` block:

```
MyDataset.DisableControls;
Try
  // Do some intensive stuff
Finally
  MyDataset.EnableControls
end;
```

Errors: Failure to call enablecontrols will prevent the controls from receiving updates. The state can be checked with TDataset.ControlsDisabled (505).

See also: TDataset.DisableControls (507), TDataset.ControlsDisabled (505)

16.23.25 TDataSet.FieldByName

Synopsis: Search a field by name.

Declaration: `function FieldByName(const FieldName: string) : TField`

Visibility: public

Description: `FieldByName` is a shortcut for Fields.FieldByName (586): it searches for the field with fieldname equalling `FieldName`. The case is performed case-insensitive. The matching field instance is returned.

Errors: If the field is not found, an EDatabaseError (458) exception will be raised.

See also: TFields.FieldByname (586), TDataset.FindField (508)

16.23.26 TDataSet.FindField

Synopsis: Find a field by name.

Declaration: `function FindField(const FieldName: string) : TField`

Visibility: public

Description: `FindField` is a shortcut for Fields.FindField (586): it searches for the field with fieldname equalling `FieldName`. The case is performed case-insensitive. The matching field instance is returned, and if no match is found, `Nil` is returned.

See also: TDataset.FieldByname (508), TFields.FindField (586)

16.23.27 TDataSet.FindFirst

Synopsis: Find the first active record (deprecated).

Declaration: `function FindFirst : Boolean;  Virtual`

Visibility: public

Description: `FindFirst` positions the cursor on the first record (taking into account filtering), and returns `True` if the cursor position was changed. This method must be implemented by descendents of `TDataset`: The implementation in `TDataset` always returns `False`, indicating that the position was not changed.

This method is deprecated, use TDataset.First (510) instead.

See also: TDataset.First (510), TDataset.FindLast (509), TDataset.FindNext (509), TDataset.FindPrior (509)

16.23.28 TDataSet.FindLast

Synopsis: Find the last active record (deprecated).

Declaration: `function FindLast : Boolean;  Virtual`

Visibility: public

Description: `FindLast` positions the cursor on the last record (taking into account filtering), and returns `True` if the cursor position was changed. This method must be implemented by descendents of `TDataset`: The implementation in `TDataset` always returns `False`, indicating that the position was not changed.

This method is deprecated, use TDataset.Last (513) instead.

See also: TDataset.Last (513), TDataset.FindFirst (509), TDataset.FindNext (509), TDataset.FindPrior (509)

16.23.29 TDataSet.FindNext

Synopsis: Find the next active record (deprecated).

Declaration: `function FindNext : Boolean;  Virtual`

Visibility: public

Description: `FindLast` positions the cursor on the next record (taking into account filtering), and returns `True` if the cursor position was changed. This method must be implemented by descendents of `TDataset`: The implementation in `TDataset` always returns `False`, indicating that the position was not changed.

This method is deprecated, use TDataset.Next (514) instead.

See also: TDataset.Next (514), TDataset.FindFirst (509), TDataset.FindLast (509), TDataset.FindPrior (509)

16.23.30 TDataSet.FindPrior

Synopsis: Find the previous active record (deprecated).

Declaration: `function FindPrior : Boolean;  Virtual`

Visibility: public

Description: `FindPrior` positions the cursor on the previous record (taking into account filtering), and returns `True` if the cursor position was changed. This method must be implemented by descendents of `TDataset`: The implementation in `TDataset` always returns `False`, indicating that the position was not changed.

This method is deprecated, use TDataset.Prior (516) instead.

See also: TDataset.Prior (516), TDataset.FindFirst (509), TDataset.FindLast (509), TDataset.FindPrior (509)

16.23.31 TDataSet.First

Synopsis: Position the dataset on the first record.

Declaration: `procedure First`

Visibility: public

Description: `First` positions the dataset on the first record. This action will trigger the TDataset.BeforeScroll (531) and TDataset.AfterScroll (531) events. After the action is completed, the TDataset.BOF (518) property will be `True`.

Errors: If the dataset is unidirectional or is closed, an EDatabaseError (458) exception will be raised.

See also: TDataset.Prior (516), TDataset.Last (513), TDataset.Next (514), TDataset.BOF (518), TDataset.BeforeScroll (531), TDataset.AfterScroll (531)

16.23.32 TDataSet.FreeBookmark

Synopsis: Free a bookmark obtained with `GetBookmark` (deprecated).

Declaration: `procedure FreeBookmark(ABookmark: TBookMark);  Virtual`

Visibility: public

Description: `FreeBookmark` must be used to free a bookmark obtained by TDataset.GetBookmark (510). It should not be used on bookmarks obtained with the TDataset.Bookmark (518) property. Both `GetBookmark` and `FreeBookmark` are deprecated. Use the `Bookmark` property instead: it uses a string type, which is automatically disposed of when the string variable goes out of scope.

See also: TDataset.GetBookmark (510), TDataset.Bookmark (518)

16.23.33 TDataSet.GetBookmark

Synopsis: Get a bookmark pointer (deprecated).

Declaration: `function GetBookmark : TBookMark;  Virtual`

Visibility: public

Description: `GetBookmark` gets a bookmark pointer to the current cursor location. The TDataset.SetBookmark (496) call can be used to return to the current record in the dataset. After use, the bookmark must be disposed of with the TDataset.FreeBookmark (510) call. The bookmark will be `Nil` if the dataset is empty or not active.

This call is deprecated. Use the TDataset.Bookmark (518) property instead to get a bookmark.

See also: TDataset.SetBookmark (496), TDataset.FreeBookmark (510), TDataset.Bookmark (518)

16.23.34 TDataSet.GetCurrentRecord

Synopsis: Copy the data for the current record in a memory buffer.

Declaration: `function GetCurrentRecord(Buffer: TRecordBuffer) : Boolean; Virtual`

Visibility: public

Description: `GetCurrentRecord` can be overridden by `TDataset` descendents to copy the data for the current record to `Buffer`. `Buffer`must point to a memory area, large enough to contain the data for the record. If the data is copied successfully to the buffer, the function returns `True`. The `TDataset` implementation is empty, and returns `False`.

See also: TDataset.ActiveBuffer (502)

16.23.35 TDataSet.GetFieldList

Synopsis: Return field instances in a list.

Declaration: `procedure GetFieldList(List: TList; const FieldNames: string)`

Visibility: public

Description: `GetfieldList` parses `FieldNames` for names of fields, and returns the field instances that match the names in `list`. `FieldNames` must be a list of field names, separated by semicolons. The list is cleared prior to filling with the requested field instances.

Errors: If `FieldNames` contains a name of a field that does not exist in the dataset, then an EDatabaseError (458) exception will be raised.

See also: TDataset.GetFieldNames (511), TDataset.FieldByName (508), TDataset.FindField (508)

16.23.36 TDataSet.GetFieldNames

Synopsis: Return a list of all available field names.

Declaration: `procedure GetFieldNames(List: TStrings)`

Visibility: public

Description: `GetFieldNames` returns in `List` the names of all available fields, one field per item in the list. The dataset must be open for this function to work correctly.

See also: TDataset.GetFieldNameList (496), TDataset.FieldByName (508), TDataset.FindField (508)

16.23.37 TDataSet.GotoBookmark

Synopsis: Jump to bookmark.

Declaration: `procedure GotoBookmark(const ABookmark: TBookMark)`

Visibility: public

Description: `GotoBookmark` positions the dataset to the bookmark position indicated by `ABookMark`. `ABookmark` is a bookmark obtained by the TDataset.GetBookmark (510) function.

This function is deprecated, use the TDataset.Bookmark (518) property instead.

Errors: if `ABookmark` does not contain a valid bookmark, then an exception may be raised.

See also: TDataset.Bookmark (518), TDataset.GetBookmark (510), TDataset.FreeBookmark (510)

16.23.38 TDataSet.Insert

Synopsis: Insert a new record at the current position.

Declaration: `procedure Insert`

Visibility: public

Description: `Insert` will insert a new record at the current position. When this function is called, any pending modifications (when the dataset already is in insert or edit mode) will be posted. After that, the BeforeInsert (528), BeforeScroll (531), OnNewRecord (534), AfterInsert (528) and AfterScroll (531) events are triggered in the order indicated here. The dataset is in the `dsInsert` state after this method is called, and the contents of the various fields can be set. To write the new record to the underlying database TDataset.Post (515) must be called.

Errors: If the dataset is read-only, calling `Insert` will result in an EDatabaseError (458).

See also: BeforeInsert (528), BeforeScroll (531), OnNewRecord (534), AfterInsert (528), AfterScroll (531), TDataset.Post (515), TDataset.Append (503)

16.23.39 TDataSet.InsertRecord

Synopsis: Insert a new record with given values.

Declaration: `procedure InsertRecord(const Values: Array of const)`

Visibility: public

Description: `InsertRecord` is not yet implemented in Free Pascal. It does nothing.

See also: TDataset.Insert (512), TDataset.SetFieldValues (496)

16.23.40 TDataSet.IsEmpty

Synopsis: Check if the dataset contains no data.

Declaration: `function IsEmpty : Boolean`

Visibility: public

Description: `IsEmpty` returns `True` if the dataset is empty, i.e. if EOF (520) and TDataset.BOF (518) are both `True`, and the dataset is not in insert mode.

See also: TDataset.EOF (520), TDataset.BOF (518), TDataset.State (524)

16.23.41 TDataSet.IsLinkedTo

Synopsis: Check whether a datasource is linked to the dataset.

Declaration: `function IsLinkedTo(ADataSource: TDataSource) : Boolean`

Visibility: public

Description: `IsLinkedTo` returns `True` if `ADatasource` is linked to this dataset, either directly (the ADatasource.Dataset" (538) points to the current dataset instance, or indirectly.

See also: TDatasource.Dataset (538)

16.23.42 TDataSet.IsSequenced

Synopsis: Is the data sequenced.

Declaration:
```
function IsSequenced : Boolean;  Virtual
```

Visibility: public

Description: `IsSequenced` indicates whether it is safe to use the TDataset.RecNo (523) property to navigate in the records of the data. By default, this property is set to `True`, but `TDataset` descendents may set this property to `False` (for instance, unidirectional datasets), in which case `RecNo` should not be used to navigate through the data.

See also: TDataset.RecNo (523)

16.23.43 TDataSet.Last

Synopsis: Navigate forward to the last record.

Declaration:
```
procedure Last
```

Visibility: public

Description: `Last` puts the cursor at the last record in the dataset, fetching more records from the underlying database if needed. It is equivalent to moving to the last record and calling TDataset.Next (514). After a call to `Last`, the TDataset.EOF (520) property will be `True`.

Calling this method will trigger the TDataset.BeforeScroll (531) and TDataset.AfterScroll (531) events.

See also: TDataset.First (510), TDataset.Next (514), TDataset.EOF (520), TDataset.BeforeScroll (531), TDataset.AfterScroll (531)

16.23.44 TDataSet.Locate

Synopsis: Locate a record based on some key values.

Declaration:
```
function Locate(const KeyFields: string; const KeyValues: Variant;
                Options: TLocateOptions) : Boolean;  Virtual
```

Visibility: public

Description: `Locate` attempts to locate a record in the dataset. There are 2 possible cases when using `Locate`.

1. `Keyvalues` is a single value. In that case, `KeyFields` is the name of the field whose value must be matched to the value in `KeyValues`
2. `Keyvalues` is a variant array. In that case, `KeyFields` must contain a list of names of fields (separated by semicolons) whose values must be matched to the values in the `KeyValues` array

The matching always happens according to the `Options` parameter. For a description of the possible values, see TLocateOption (450).

If a record is found that matches the criteria, then the `locate` operation positions the cursor on this record, and returns `True`. If no record is found to match the criteria, `False` is returned, and the position of the cursor is unchanged.

The implementation in `TDataset` always returns `False`. It is up to `TDataset` descendents to implement this method and return an appropriate value.

See also: TDataset.Find (496), TDataset.Lookup (514), TLocateOption (450)

16.23.45 TDataSet.Lookup

Synopsis: Search for a record and return matching values.

Declaration:
```
function Lookup(const KeyFields: string; const KeyValues: Variant;
                const ResultFields: string) : Variant;  Virtual
```

Visibility: public

Description: `Lookup` always returns `Null` in `TDataset`. Descendents of `TDataset` can override this method to call TDataset.Locate (513) to locate the record with fields `KeyFields` matching `KeyValues` and then to return the values of the fields in `ResultFields`. If `ResultFields` contains more than one fieldname (separated by semicolons), then the function returns an array. If there is only 1 fieldname, the value is returned directly.

Errors: If the dataset is unidirectional, then a EDatabaseError (458) exception will be raised.

See also: TDataset.Locate (513)

16.23.46 TDataSet.MoveBy

Synopsis: Move the cursor position.

Declaration:
```
function MoveBy(Distance: LongInt) : LongInt
```

Visibility: public

Description: `MoveBy` moves the current record pointer with `Distance` positions. `Distance` may be a positive number, in which case the cursor is moved forward, or a negative number, in which case the cursor is moved backward. The move operation will stop as soon as the beginning or end of the data is reached. The TDataset.BeforeScroll (531) and TDataset.AfterScroll (531) events are triggered (once) when this method is called. The function returns the distance which was actually moved by the cursor.

Errors: A negative distance will result in an EDatabaseError (458) exception on unidirectional datasets.

See also: TDataset.RecNo (523), TDataset.BeforeScroll (531), TDataset.AfterScroll (531)

16.23.47 TDataSet.Next

Synopsis: Go to the next record in the dataset.

Declaration:
```
procedure Next
```

Visibility: public

Description: `Next` positions the cursor on the next record in the dataset. It is equivalent to a `MoveBy(1)` operation. Calling this method triggers the TDataset.BeforeScroll (531) and TDataset.AfterScroll (531) events. If the dataset is located on the last known record (EOF (520) is true), then no action is performed, and the events are not triggered.

Errors: Calling this method on a closed dataset will result in an EDatabaseError (458) exception.

See also: TDataset.MoveBy (514), TDataset.Prior (516), TDataset.Last (513), TDataset.BeforeScroll (531), TDataset.AfterScroll (531), TDataset.EOF (520)

16.23.48 TDataSet.Open

Synopsis: Activate the dataset: Fetch data into memory.

Declaration: `procedure Open`

Visibility: public

Description: `Open` must be used to make the `TDataset` Active. It does nothing if the dataset is already active. `Open` initializes the `TDataset` and brings the dataset in a browsable state:

Effectively the following happens:

1. The BeforeOpen event is triggered.
2. The descendents `InternalOpen` method is called to actually fetch data and initialize field-defs and field instances.
3. BOF (518)is set to `True`
4. Internal buffers are allocated and filled with data
5. If the dataset is empty, EOF (520) is set to true
6. State (524) is set to `dsBrowse`
7. The AfterOpen (527) event is triggered

Errors: If the descendent class cannot fetch the data, or the data does not match the field definitions present in the dataset, then an exception will be raised.

See also: TDataset.Active (526), TDataset.State (524), TDataset.BOF (518), TDataset.EOF (520), TDataset.BeforeOpen (527), TDataset.AfterOpen (527)

16.23.49 TDataSet.Post

Synopsis: Post pending edits to the database.

Declaration: `procedure Post; Virtual`

Visibility: public

Description: `Post` attempts to save pending edits when the dataset is in one of the edit modes: that is, after a Insert (512), Append (503) or TDataset.Edit (507) operation. The changes will be committed to memory - and usually immediately to the underlying database as well. Prior to saving the data to memory, it will check some constraints: in `TDataset`, the presence of a value for all required fields is checked. if for a required field no value is present, an exception will be raised. A call to `Post` results in the triggering of the BeforePost (529), AfterPost (529) events. After the call to `Past`, the State (524) of the dataset is again `dsBrowse`, i.e. the dataset is again in browse mode.

Errors: Invoking the `post` method when the dataset is not in one of the editing modes (dsEditModes (439)) will result in an EdatabaseError (458) exception. If an exception occurs during the save operation, the OnPostError (534) event is triggered to handle the error.

See also: Insert (512), Append (503), Edit (507), OnPostError (534), BeforePost (529), AfterPost (529), State (524)

16.23.50 TDataSet.Prior

Synopsis: Go to the previous record.

Declaration: `procedure Prior`

Visibility: public

Description: `Prior` moves the cursor to the previous record. It is equivalent to a `MoveBy(-1)` operation. Calling this method triggers the TDataset.BeforeScroll (531) and TDataset.AfterScroll (531) events. If the dataset is located on the first record, (BOF (518) is true) then no action is performed, and the events are not triggered.

Errors: Calling this method on a closed dataset will result in an EDatabaseError (458) exception.

See also: TDataset.MoveBy (514), TDataset.Next (514), TDataset.First (510), TDataset.BeforeScroll (531), TDataset.AfterScroll (531), TDataset.BOF (518)

16.23.51 TDataSet.Refresh

Synopsis: Refresh the records in the dataset.

Declaration: `procedure Refresh`

Visibility: public

Description: `Refresh` posts any pending edits, and refetches the data in the dataset from the underlying database, and attempts to reposition the cursor on the same record as it was. This operation is not supported by all datasets, and should be used with care. The repositioning may not always succeed, in which case the cursor will be positioned on the first record in the dataset. This is in particular true for unidirectional datasets. Calling `Refresh` results in the triggering of the BeforeRefresh (532) and AfterRefresh (532) events.

Errors: Refreshing may fail if the underlying dataset descendent does not support it.

See also: TDataset.Close (505), TDataset.Open (515), BeforeRefresh (532), AfterRefresh (532)

16.23.52 TDataSet.Resync

Synopsis: Resynchronize the data buffer.

Declaration: `procedure Resync(Mode: TResyncMode);  Virtual`

Visibility: public

Description: `Resync` refetches the records around the cursor position. It should not be used by application code, instead TDataset.Refresh (516) should be used. The `Resync` parameter indicates how the buffers should be refreshed.

See also: TDataset.Refresh (516)

16.23.53 TDataSet.SetFields

Synopsis: Set a number of field values at once.

Declaration: `procedure SetFields(const Values: Array of const)`

Visibility: public

Description: `SetFields` sets the values of the fields with the corresponding values in the array. It starts with the first field in the TDataset.Fields (524) property, and works it's way down the array.

Errors: If the dataset is not in edit mode, then an EDatabaseError (458) exception will be raised. If there are more values than fields, an `EListError` exception will be raised.

See also: TDataset.Fields (524)

16.23.54 TDataSet.Translate

Synopsis: Transliterate a buffer.

Declaration:
```
function Translate(Src: PChar; Dest: PChar; ToOem: Boolean) : Integer
                        ;  Virtual
```

Visibility: public

Description: `Translate` is called for all string fields for which the TStringField.Transliterate (642) property is set to `True`. The `toOEM` parameter is set to `True` if the transliteration must happen from the used codepage to the codepage used for storage, and if it is set to `False` then the transliteration must happen from the native codepage to the storage codepage. This call must be overridden by descendents of `TDataset` to provide the necessary transliteration: `TDataset` just copies the contents of the `Src` buffer to the `Dest` buffer. The result must be the number of bytes copied to the destination buffer.

Errors: No checks are performed on the bufffers.

See also: TStringField.Transliterate (642)

16.23.55 TDataSet.UpdateCursorPos

Synopsis: Update cursor position.

Declaration:
```
procedure UpdateCursorPos
```

Visibility: public

Description: `UpdateCursorPos` should not be used in application code. It is used to ensure that the logical cursor position is the correct (physical) position.

See also: TDataset.Refresh (516)

16.23.56 TDataSet.UpdateRecord

Synopsis: Indicate that the record contents have changed.

Declaration:
```
procedure UpdateRecord
```

Visibility: public

Description: `UpdateRecord` notifies controls that the contents of the current record have changed. It triggers the event. This should never be called by application code, and is intended only for descendents of `TDataset`.

See also: OnUpdateRecord (496)

16.23.57 TDataSet.UpdateStatus

Synopsis: Get the update status for the current record.

Declaration: `function UpdateStatus : TUpdateStatus;  Virtual`

Visibility: public

Description: `UpdateStatus` always returns `usUnModified` in the `TDataset` implementation. Descendent classes should override this method to indicate the status for the current record in case they support cached updates: the function should return the status of the current record: has the record been locally inserted, modified or deleted, or none of these. `UpdateStatus` is not used in `TDataset` itself, but is provided so applications have a unique API to work with datasets that have support for cached updates.

16.23.58 TDataSet.BlockReadSize

Synopsis: Number of records to read.

Declaration: `Property BlockReadSize : Integer`

Visibility: public

Access: Read,Write

Description: `BlockReadSize` can be set to a positive number to prevent the dataset from sending notifications to DB-Aware controls while scrolling through the data. Setting it to zero will re-enable sending of notifications, as will putting the dataset in another state (edit etc.).

See also: EnableControls (438), DisableControls (438)

16.23.59 TDataSet.BOF

Synopsis: Is the cursor at the beginning of the data (on the first record).

Declaration: `Property BOF : Boolean`

Visibility: public

Access: Read

Description: `BOF` returns `True` if the first record is the first record in the dataset, `False` otherwise. It will always be `True` if the dataset is just opened, or after a call to TDataset.First (510). As soon as TDataset.Next (514) is called, `BOF` will no longer be true.

See also: TDataset.EOF (520), TDataset.Next (514), TDataset.First (510)

16.23.60 TDataSet.Bookmark

Synopsis: Get or set the current cursor position.

Declaration: `Property Bookmark : TBookMark`

Visibility: public

Access: Read,Write

Description: `Bookmark` can be read to obtain a bookmark to the current position in the dataset. The obtained value can be used to return to current position at a later stage. Writing the `Bookmark` property with a value previously obtained like this, will reposition the dataset on the same position as it was when the property was read.

This is often used when scanning all records, like this:

```
Var
  B : TBookmark;

begin
  With MyDataset do
    begin
    B:=Bookmark;
    DisableControls;
    try
      First;
      While Not EOF do
        begin
        DoSomething;
        Next;
        end;
    finally
      EnableControls;
      Bookmark:=B;
    end;
    end;
```

At the end of this code, the dataset will be positioned on the same record as when the code was started. The TDataset.DisableControls (507) and TDataset.EnableControls (508) calls prevent the controls from receiving update notifications as the dataset scrolls through the records, thus reducing flicker on the screen.

Note that bookmarks become invalid as soon as the dataset closes. A call to refresh may also destroy the bookmarks.

See also: TDataset.DisableControls (507), TDataset.EnableControls (508)

16.23.61 TDataSet.CanModify

Synopsis: Can the data in the dataset be modified.

Declaration: `Property CanModify : Boolean`

Visibility: public

Access: Read

Description: `CanModifiy` indicates whether the dataset allows editing. `Unidirectional` datasets do not allow editing. Descendent datasets can impose additional conditions under which the data can not be modified (read-only datasets, for instance). If the `CanModify` property is `False`, then the edit, append or insert methods will fail.

See also: TDataset.Insert (512), TDataset.Append (503), TDataset.Delete (506), Tdataset.Edit (507)

16.23.62 TDataSet.DataSource

Synopsis: Datasource this dataset is connected to.

Declaration: `Property DataSource : TDataSource`

Visibility: public

Access: Read

Description: `Datasource` is the datasource this dataset is connected to, and from which it can get values for parameters. In `TDataset`, the `Datasource` property is not used, and is always `Nil`. It is up to descendent classes that actually support a datasource to implement getter and setter routines for the `Datasource` property.

See also: TDatasource (536)

16.23.63 TDataSet.DefaultFields

Synopsis: Is the dataset using persistent fields or not.

Declaration: `Property DefaultFields : Boolean`

Visibility: public

Access: Read

Description: `DefaultFields` is `True` if the fields were generated dynamically when the dataset was opened. If it is `False` then the field instances are persistent, i;e. they were created at design time with the fields editor. If `DefaultFields` is `True`, then for each item in the TDataset.FieldDefs (521) property, a field instance is created. These fields instances are freed again when the dataset is closed.

If `DefaultFields` is `False`, then there may be less field instances than there are items in the `FieldDefs` property. This can be the case for instance when opening a DBF file at runtime which has more fields than the file used at design time.

See also: TDataset.FieldDefs (521), TDataset.Fields (524), TField (549)

16.23.64 TDataSet.EOF

Synopsis: Indicates whether the last record has been reached.

Declaration: `Property EOF : Boolean`

Visibility: public

Access: Read

Description: `EOF` is `True` if the cursor is on the last record in the dataset, and no more records are available. It is also `True` for an empty dataset. The `EOF` property will be set to `True` in the following cases:

1. The cursor is on the last record, and the TDataset.Next (514) method is called.
2. The TDataset.Last (513) method is called (which is equivalent to moving to the last record and calling TDataset.Next (514)).
3. The dataset is empty when opened.

In all other cases, `EOF` is `False`. Note: when the cursor is on the last-but-one record, and `Next` is called (moving the cursor to the last record), `EOF` will not yet be `True`. Only if both the cursor is on the last record **and**`Next` is called, will `EOF` become `True`.

This means that the following loop will stop after the last record was visited:

```
With MyDataset do
  While not EOF do
    begin
    DoSomething;
    Next;
    end;
```

See also: TDataset.BOF (518), TDataset.Next (514), TDataset.Last (513), TDataset.IsEmpty (512)

16.23.65 TDataSet.FieldCount

Synopsis: Number of fields.

Declaration: `Property FieldCount : LongInt`

Visibility: public

Access: Read

Description: `FieldCount` is the same as Fields.Count (588), i.e. the number of fields. For a dataset with persistent fields (when DefaultFields (520) is `False`) then this number will be always the same every time the dataset is opened. For a dataset with dynamically created fields, the number of fields may be different each time the dataset is opened.

See also: TFields (584)

16.23.66 TDataSet.FieldDefs

Synopsis: Definitions of available fields in the underlying database.

Declaration: `Property FieldDefs : TFieldDefs`

Visibility: public

Access: Read,Write

Description: `FieldDefs` is filled by the `TDataset` descendent when the dataset is opened. It represents the fields as they are returned by the particular database when the data is initially fetched from the engine. If the dataset uses dynamically created fields (when DefaultFields (520) is `True`), then for each item in this list, a field i instance will be created with default properties available in the field definition. If the dataset uses persistent fields, then the fields in the field list will be checked against the items in the `FieldDefs` property. If no matching item is found for a persistent field, then an exception will be raised. Items that exist in the fielddefs property but for which there is no matching field instance, are ignored.

See also: TDataset.Open (515), TDataset.DefaultFields (520), TDataset.Fields (524)

16.23.67 TDataSet.Found

Synopsis: Check success of one of the `Find` methods.

Declaration: `Property Found : Boolean`

Visibility: public

Access: Read

Description: `Found` is `True` if the last of one of the TDataset.FindFirst (509),TDataset.FindLast (509),TDataset.FindNext (509) or TDataset.FindPrior (509) operations was successful.

See also: TDataset.FindFirst (509), TDataset.FindLast (509), TDataset.FindNext (509), TDataset.FindPrior (509)

16.23.68 TDataSet.Modified

Synopsis: Was the current record modified ?

Declaration: `Property Modified : Boolean`

Visibility: public

Access: Read

Description: `Modified` is `True` if the current record was modified after a call to Tdataset.Edit (507) or Tdataset.Insert (512). It becomes `True` if a value was written to one of the fields of the dataset.

See also: Tdataset.Edit (507), TDataset.Insert (512), TDataset.Append (503), TDataset.Cancel (504), TDataset.Post (515)

16.23.69 TDataSet.IsUniDirectional

Synopsis: Is the dataset unidirectional (i.e. forward scrolling only).

Declaration: `Property IsUniDirectional : Boolean`

Visibility: public

Access: Read

Description: `IsUniDirectional` is `True` if the dataset is unidirectional. By default it is `False`, i.e. scrolling backwards is allowed. If the dataset is unidirectional, then any attempt to scroll backwards (using one of TDataset.Prior (516) or TDataset.Last (513)), random positioning of the cursor, editing or filtering will result in an EDatabaseError (458). Unidirectional datasets are also not suitable for display in a grid, as they have only 1 record in memory at any given time: they are only useful for performing an action on all records:

```
With MyDataset do
  While not EOF do
    begin
    DoSomething;
    Next;
    end;
```

See also: TDataset.Prior (516), TDataset.Next (514)

16.23.70 TDataSet.RecordCount

Synopsis: Number of records in the dataset.

Declaration: `Property RecordCount : LongInt`

Visibility: public

Access: Read

Description: `RecordCount` is the number of records in the dataset. This number is not necessarily equal to the number of records returned by a query. For optimization purposes, a `TDataset` descendent may choose not to fetch all records from the database when the dataset is opened. If this is the case, then the `RecordCount` will only reflect the number of records that have actually been fetched at the current time, and therefor the value will change as more records are fetched from the database.

Only when `Last` has been called (and the dataset has been forced to fetch all records returned by the database), will the value of `RecordCount` be equal to the number of records returned by the query.

In general, datasets based on in-memory data or flat files, will return the correct number of records in `RecordCount`.

See also: TDataset.RecNo (523)

16.23.71 TDataSet.RecNo

Synopsis: Current record number.

Declaration: `Property RecNo : LongInt`

Visibility: public

Access: Read,Write

Description: `RecNo` returns the current position in the dataset. It can be written to set the cursor to the indicated position. This property must be implemented by `TDataset` descendents, for `TDataset` the property always returns -1.

This property should not be used if exact positioning is required. it is inherently unreliable.

See also: TDataset.RecordCount (523)

16.23.72 TDataSet.RecordSize

Synopsis: Size of the record in memory.

Declaration: `Property RecordSize : Word`

Visibility: public

Access: Read

Description: `RecordSize` is the total size of the memory buffer used for the records. This property returns always 0 in the `TDataset` implementation. Descendent classes should implement this property. Note that this property does not necessarily reflect the actual data size for the records. that may be more or less, depending on how the `TDataset` descendent manages it's data.

See also: TField.Datasize (562), TDataset.RecordCount (523), TDataset.RecNo (523)

16.23.73 TDataSet.SparseArrays

Declaration: `Property SparseArrays : Boolean`

Visibility: public

Access: Read,Write

16.23.74 TDataSet.State

Synopsis: Current operational state of the dataset.

Declaration: `Property State : TDataSetState`

Visibility: public

Access: Read

Description: `State` determines the current operational state of the dataset. During it's lifetime, the dataset is in one of many states, depending on which operation is currently in progress:

- If a dataset is closed, the `State` is `dsInactive`.
- As soon as it is opened, it is in `dsBrowse` mode, and remains in this state while changing the cursor position.
- If the `Edit` or `Insert` or `Append` methods is called, the `State` changes to `dsEdit` or `dsInsert`, respectively.
- As soon as edits have been posted or cancelled, the state is again `dsBrowse`.
- Closing the dataset sets the state again to `dsInactive`.

There are some other states, mainly connected to internal operations, but which can become visible in some of the dataset's events.

See also: TDataset.Active (526), TDataset.Edit (507), TDataset.Insert (512), TDataset.Append (503), TDataset.Post (515), TDataset.Cancel (504)

16.23.75 TDataSet.Fields

Synopsis: Indexed access to the fields of the dataset.

Declaration: `Property Fields : TFields`

Visibility: public

Access: Read

Description: `Fields` provides access to the fields of the dataset. It is of type TFields (584) and therefore gives indexed access to the fields, but also allows other operations such as searching for fields based on their names or getting a list of fieldnames.

See also: TFieldDefs (581), TField (549)

16.23.76 TDataSet.FieldValues

Synopsis: Access to field values based on the field names.

Declaration: `Property FieldValues[FieldName: string]: Variant; default`

Visibility: public

Access: Read,Write

Description: `FieldValues` provides array-like access to the values of the fields, based on the names of the fields. The value is read or written as a variant type. It is equivalent to the following:

```
FieldByName(FieldName).AsVariant
```

It can be read as well as written.

See also: TFields.FieldByName (586)

16.23.77 TDataSet.Filter

Synopsis: Filter to apply to the data in memory.

Declaration: `Property Filter : string`

Visibility: public

Access: Read,Write

Description: `Filter` is not implemented by `TDataset`. It is up to descendent classes to implement actual filtering: the filtering happens on in-memory data, and is not applied on the database level. (in particular: setting the filter property will in no way influence the WHERE clause of an SQL-based dataset).

In general, the `filter` property accepts a SQL-like syntax usually encountered in the WHERE clause of an SQL SELECT statement.

The filter is only applied if the `Filtered` property is set to `True`. If the `Filtered` property is `False`, the `Filter` property is ignored.

See also: TDataset.Filtered (525), TDataset.FilterOptions (526)

16.23.78 TDataSet.Filtered

Synopsis: Is the filter active or not.

Declaration: `Property Filtered : Boolean`

Visibility: public

Access: Read,Write

Description: `Filtered` determines whether the filter condition in TDataset.Filter (525) is applied or not. The filter is only applied if the `Filtered` property is set to `True`. If the `Filtered` property is `False`, the `Filter` property is ignored.

See also: TDataset.Filter (525), TDataset.FilterOptions (526)

16.23.79 TDataSet.FilterOptions

Synopsis: Options to apply when filtering.

Declaration: `Property FilterOptions : TFilterOptions`

Visibility: public

Access: Read,Write

Description: `FilterOptions` determines what options should be taken into account when applying the filter in TDataset.Filter (525), such as case-sensitivity or whether to treat an asterisk as a wildcard: By default, an asterisk (*) at the end of a literal string in the filter expression is treated as a wildcard. When `FilterOptions` does not include `foNoPartialCompare`, strings that have an asterisk at the end, indicate a partial string match. In that case, the asterisk matches any number of characters. If `foNoPartialCompare` is included in the options, the asterisk is regarded as a regular character.

See also: TDataset.Filter (525), TDataset.FilterOptions (526)

16.23.80 TDataSet.Active

Synopsis: Is the dataset open or closed.

Declaration: `Property Active : Boolean`

Visibility: public

Access: Read,Write

Description: `Active` is `True` if the dataset is open, and `False` if it is closed (TDataset.State (524) is then `dsInactive`). Setting the `Active` property to `True` is equivalent to calling TDataset.Open (515), setting it to `False` is equivalent to calling TDataset.Close (505)

See also: TDataset.State (524), TDataset.Open (515), TDataset.Close (505)

16.23.81 TDataSet.AutoCalcFields

Synopsis: How often should the value of calculated fields be calculated.

Declaration: `Property AutoCalcFields : Boolean`

Visibility: public

Access: Read,Write

Description: `AutoCalcFields` is by default `true`, meaning that the values of calculated fields will be computed in the following cases:

- When the dataset is opened
- When the dataset is put in edit mode
- When a data field changed

When `AutoCalcFields` is `False`, then the calculated fields are called whenever

- The dataset is opened
- The dataset is put in edit mode

Both proper calculated fields and lookup fields are computed. Calculated fields are computed through the TDataset.OnCalcFields (532) event.

See also: TField.FieldKind (569), TDataset.OnCalcFields (532)

16.23.82 TDataSet.BeforeOpen

Synopsis: Event triggered before the dataset is opened.

Declaration: `Property BeforeOpen : TDataSetNotifyEvent`

Visibility: public

Access: Read,Write

Description: `BeforeOpen` is triggered before the dataset is opened. No actions have been performed yet when this event is called, and the dataset is still in `dsInactive` state. It can be used to set parameters and options that influence the opening process. If an exception is raised during the event handler, the dataset remains closed.

See also: TDataset.AfterOpen (527), TDataset.State (524)

16.23.83 TDataSet.AfterOpen

Synopsis: Event triggered after the dataset is opened.

Declaration: `Property AfterOpen : TDataSetNotifyEvent`

Visibility: public

Access: Read,Write

Description: `AfterOpen` is triggered after the dataset is opened. The dataset has fetched its data and is in `dsBrowse` state when this event is triggered. If the dataset is not empty, then a TDataset.AfterScroll (531) event will be triggered immediately after the `AfterOpen` event. If an exception is raised during the event handler, the dataset remains open, but the `AfterScroll` event will not be triggered.

See also: TDataset.AfterOpen (527), TDataset.State (524), TDataset.AfterScroll (531)

16.23.84 TDataSet.BeforeClose

Synopsis: Event triggered before the dataset is closed.

Declaration: `Property BeforeClose : TDataSetNotifyEvent`

Visibility: public

Access: Read,Write

Description: `BeforeClose` is triggered before the dataset is closed. No actions have been performed yet when this event is called, and the dataset is still in `dsBrowse` state or one of the editing states. It can be used to prevent closing of the dataset, for instance if there are pending changes not yet committed to the database. If an exception is raised during the event handler, the dataset remains opened.

See also: TDataset.AfterClose (527), TDataset.State (524)

16.23.85 TDataSet.AfterClose

Synopsis: Event triggered after the dataset is closed.

Declaration: `Property AfterClose : TDataSetNotifyEvent`

Visibility: public

Access: Read,Write

Description: `AfterOpen` is triggered after the dataset is opened. The dataset has discarded its data and has cleaned up it's internal memory structures. It is in `dsInactive` state when this event is triggered.

See also: TDataset.BeforeClose (527), TDataset.State (524)

16.23.86 TDataSet.BeforeInsert

Synopsis: Event triggered before the dataset is put in insert mode.

Declaration: `Property BeforeInsert : TDataSetNotifyEvent`

Visibility: public

Access: Read,Write

Description: `BeforeInsert` is triggered at the start of the TDataset.Append (503) or TDataset.Insert (512) methods. The dataset is still in `dsBrowse` state when this event is triggered. If an exception is raised in the `BeforeInsert` event handler, then the dataset will remain in `dsBrowse` state, and the append or insert operation is cancelled.

See also: TDataset.AfterInsert (528), TDataset.Append (503), TDataset.Insert (512)

16.23.87 TDataSet.AfterInsert

Synopsis: Event triggered after the dataset is put in insert mode.

Declaration: `Property AfterInsert : TDataSetNotifyEvent`

Visibility: public

Access: Read,Write

Description: `AfterInsert` is triggered after the dataset has finished putting the dataset in `dsInsert` state and it has initialized the new record buffer. This event can be used e.g. to set initial field values. After the `Afterinsert` event, the TDataset.AfterScroll (531) event is still triggered. Raising an exception in the `AfterInsert` event, will prevent the `AfterScroll` event from being triggered, but does not undo the insert or append operation.

See also: TDataset.BeforeInsert (528), TDataset.AfterScroll (531), TDataset.Append (503), TDataset.Insert (512)

16.23.88 TDataSet.BeforeEdit

Synopsis: Event triggered before the dataset is put in edit mode.

Declaration: `Property BeforeEdit : TDataSetNotifyEvent`

Visibility: public

Access: Read,Write

Description: `BeforeEdit` is the triggered at the start of the TDataset.Edit (507) method. The dataset is still in `dsBrowse` state when this event is triggered. If an exception is raised in the `BeforeEdit` event handler, then the dataset will remain in `dsBrowse` state, and the edit operation is cancelled.

See also: TDataset.AfterEdit (529), TDataset.Edit (507), TDataset.State (524)

16.23.89 TDataSet.AfterEdit

Synopsis: Event triggered after the dataset is put in edit mode.

Declaration: `Property AfterEdit : TDataSetNotifyEvent`

Visibility: public

Access: Read,Write

Description: `AfterEdit` is triggered after the dataset has finished putting the dataset in `dsEdit` state and it has initialized the edit buffer for the record. Raising an exception in the `AfterEdit` event does not undo the edit operation.

See also: TDataset.BeforeEdit (528), TDataset.Edit (507), TDataset.State (524)

16.23.90 TDataSet.BeforePost

Synopsis: Event called before changes are posted to the underlying database.

Declaration: `Property BeforePost : TDataSetNotifyEvent`

Visibility: public

Access: Read,Write

Description: `BeforePost` is triggered at the start of the TDataset.Post (515) method, when the dataset is still in one of the edit states (`dsEdit`,`dsInsert`). If the dataset was not in an edit state when `Post` is called, the `BeforePost` event is not triggered. This event can be used to supply values for required fields that have no value yet (the `Post` operation performs the check on required fields only after this event), or it can be used to abort the post operation: if an exception is raised during the `BeforePost` operation, the posting operation is cancelled, and the dataset remains in the editing state it was in before the post operation.

See also: TDataset.post (515), TDataset.AfterPost (529), TDataset.State (524)

16.23.91 TDataSet.AfterPost

Synopsis: Event called after changes have been posted to the underlying database.

Declaration: `Property AfterPost : TDataSetNotifyEvent`

Visibility: public

Access: Read,Write

Description: `AfterPost` is triggered when the TDataset.Post (515) operation was successfully completed, and the dataset is again in `dsBrowse` state. If an error occurred during the post operation, then the `AfterPost` event is not called, but the TDataset.OnPostError (534) event is triggered instead.

See also: TDataset.BeforePost (529), TDataset.Post (515), TDataset.State (524), TDataset.OnPostError (534)

16.23.92 TDataSet.BeforeCancel

Synopsis: Event triggered before a Cancel operation.

Declaration: `Property BeforeCancel : TDataSetNotifyEvent`

Visibility: public

Access: Read,Write

Description: `BeforeCancel` is triggered at the start of the TDataset.Cancel (504) operation, when the state is still one of the editing states (`dsEdit`,`dsInsert`). The event handler can be used to abort the cancel operation: if an exception is raised during the event handler, then the cancel operation stops. If the dataset was not in one of the editing states when the `Cancel` method was called, then the event is not triggered.

See also: TDataset.AfterCancel (530), TDataset.Cancel (504), TDataset.State (524)

16.23.93 TDataSet.AfterCancel

Synopsis: Event triggered after a Cancel operation.

Declaration: `Property AfterCancel : TDataSetNotifyEvent`

Visibility: public

Access: Read,Write

Description: `AfterCancel` is triggered when the TDataset.Cancel (504) operation was successfully completed, and the dataset is again in `dsBrowse` state.

See also: TDataset.BeforeCancel (530), TDataset.Cancel (504), TDataset.State (524)

16.23.94 TDataSet.BeforeDelete

Synopsis: Event triggered before a Delete operation.

Declaration: `Property BeforeDelete : TDataSetNotifyEvent`

Visibility: public

Access: Read,Write

Description: `BeforeDelete` is triggered at the start of the TDataset.Delete (506) operation, when the dataset is still in `dsBrowse` state. The event handler can be used to abort the delete operation: if an exception is raised during the event handler, then the delete operation stops. The event is followed by a TDataset.BeforeScroll (531) event. If the dataset was in insert mode when the `Delete` method was called, then the event will not be called, as TDataset.Cancel (504) is called instead.

See also: TDataset.AfterDelete (531), TDataset.Delete (506), TDataset.BeforeScroll (531), TDataset.Cancel (504), TDataset.State (524)

16.23.95 TDataSet.AfterDelete

Synopsis: Event triggered after a successful Delete operation.

Declaration: `Property AfterDelete : TDataSetNotifyEvent`

Visibility: public

Access: Read,Write

Description: `AfterDelete` is triggered after the successful completion of the TDataset.Delete (506) operation, when the dataset is again in `dsBrowse` state. The event is followed by a TDataset.AfterScroll (531) event.

See also: TDataset.BeforeDelete (530), TDataset.Delete (506), TDataset.AfterScroll (531), TDataset.State (524)

16.23.96 TDataSet.BeforeScroll

Synopsis: Event triggered before the cursor changes position.

Declaration: `Property BeforeScroll : TDataSetNotifyEvent`

Visibility: public

Access: Read,Write

Description: `BeforeScroll` is triggered before the cursor changes position. This can happen with one of the navigation methods: TDataset.Next (514), TDataset.Prior (516), TDataset.First (510), TDataset.Last (513), but also with two of the editing operations:TDataset.Insert (512) and TDataset.Delete (506). Raising an exception in this event handler aborts the operation in progress.

See also: TDataset.AfterScroll (531), TDataset.Next (514), TDataset.Prior (516), TDataset.First (510), TDataset.Last (513), TDataset.Insert (512), TDataset.Delete (506)

16.23.97 TDataSet.AfterScroll

Synopsis: Event triggered after the cursor has changed position.

Declaration: `Property AfterScroll : TDataSetNotifyEvent`

Visibility: public

Access: Read,Write

Description: `AfterScroll` is triggered after the cursor has changed position. This can happen with one of the navigation methods: TDataset.Next (514), TDataset.Prior (516), TDataset.First (510), TDataset.Last (513), but also with two of the editing operations:TDataset.Insert (512) and TDataset.Delete (506) and after the dataset was opened. It is suitable for displaying status information or showing a value that needs to be calculated for each record.

See also: TDataset.AfterScroll (531), TDataset.Next (514), TDataset.Prior (516), TDataset.First (510), TDataset.Last (513), TDataset.Insert (512), TDataset.Delete (506), TDataset.Open (515)

16.23.98 TDataSet.BeforeRefresh

Synopsis: Event triggered before the data is refreshed.

Declaration: `Property BeforeRefresh : TDataSetNotifyEvent`

Visibility: public

Access: Read,Write

Description: `BeforeRefresh` is triggered at the start of the TDataset.Refresh (516) method, after the dataset has been put in browse mode. If the dataset cannot be put in browse mode, the `BeforeRefresh` method will not be triggered. If an exception is raised during the `BeforeRefresh` method, then the refresh method is cancelled and the dataset remains in the `dsBrowse` state.

See also: TDataset.Refresh (516), TDataset.AfterRefresh (532), TDataset.State (524)

16.23.99 TDataSet.AfterRefresh

Synopsis: Event triggered after the data has been refreshed.

Declaration: `Property AfterRefresh : TDataSetNotifyEvent`

Visibility: public

Access: Read,Write

Description: `AfterRefresh` is triggered at the end of the TDataset.Refresh (516) method, after the dataset has refreshed its data and is again in `dsBrowse` state. This event can be used to react on changes in data in the current record

See also: TDataset.Refresh (516), TDataset.State (524), TDataset.BeforeRefresh (532)

16.23.100 TDataSet.OnCalcFields

Synopsis: Event triggered when values for calculated fields must be computed.

Declaration: `Property OnCalcFields : TDataSetNotifyEvent`

Visibility: public

Access: Read,Write

Description: `OnCalcFields` is triggered whenever the dataset needs to (re)compute the values of any calculated fields in the dataset. It is called very often, so this event should return as quickly as possible. Only the values of the calculated fields should be set, no methods of the dataset that change the data or cursor position may be called during the execution of this event handler. The frequency with which this event is called can be controlled through the TDataset.AutoCalcFields (526) property. Note that the value of lookup fields does not need to be calculated in this event, their value is computed automatically before this event is triggered.

See also: TDataset.AutoCalcFields (526), TField.Kind (549)

16.23.101 TDataSet.OnDeleteError

Synopsis: Event triggered when a delete operation fails.

Declaration: `Property OnDeleteError : TDataSetErrorEvent`

Visibility: public

Access: Read,Write

Description: `OnDeleteError` is triggered when the TDataset.Delete (506) method fails to delete the record in the underlying database. The event handler can be used to indicate what the response to the failed delete should be. To this end, it gets the exception object passed to it (parameter `E`), and it can examine this object to return an appropriate action in the `DataAction` parameter. The following responses are supported:

daFailThe operation should fail (an exception will be raised).

daAbortThe operation should be aborted (edits are undone, and an EAbort exception is raised).

daRetryRetry the operation.

For more information, see also the description of the TDatasetErrorEvent (443) event handler type.

See also: TDatasetErrorEvent (443), TDataset.Delete (506), TDataset.OnEditError (533), TDataset.OnPostError (534)

16.23.102 TDataSet.OnEditError

Synopsis: Event triggered when an edit operation fails.

Declaration: `Property OnEditError : TDataSetErrorEvent`

Visibility: public

Access: Read,Write

Description: `OnEditError` is triggered when the TDataset.Edit (507) method fails to put the dataset in edit mode because the underlying database engine reported an error. The event handler can be used to indicate what the response to the failed edit operation should be. To this end, it gets the exception object passed to it (parameter `E`), and it can examine this object to return an appropriate action in the `DataAction` parameter. The following responses are supported:

daFailThe operation should fail (an exception will be raised).

daAbortThe operation should be aborted (edits are undone, and an EAbort exception is raised).

daRetryRetry the operation.

For more information, see also the description of the TDatasetErrorEvent (443) event handler type.

See also: TDatasetErrorEvent (443), TDataset.Edit (507), TDataset.OnDeleteError (533), TDataset.OnPostError (534)

16.23.103 TDataSet.OnFilterRecord

Synopsis: Event triggered to filter records.

Declaration: `Property OnFilterRecord : TFilterRecordEvent`

Visibility: public

Access: Read,Write

Description: `OnFilterRecord` can be used to provide event-based filtering for datasets that support it. This event is only triggered when the Tdataset.Filtered (525) property is set to `True`. The event handler should set the `Accept` parameter to `True` if the current record should be accepted, or to `False` if it should be rejected. No methods that change the state of the dataset may be used during this event, and calculated fields or lookup field values are not yet available.

See also: TDataset.Filter (525), TDataset.Filtered (525), TDataset.state (524)

16.23.104 TDataSet.OnNewRecord

Synopsis: Event triggered when a new record is created.

Declaration: `Property OnNewRecord : TDataSetNotifyEvent`

Visibility: public

Access: Read,Write

Description: `OnNewRecord` is triggered by the TDataset.Append (503) or TDataset.Insert (512) methods when the buffer for the new record's data has been allocated. This event can be used to set default value for some of the fields in the dataset. If an exception is raised during this event handler, the operation is cancelled and the dataset is put again in browse mode (TDataset.State (524) is again `dsBrowse`).

See also: TDataset.Append (503), TDataset.Insert (512), TDataset.State (524)

16.23.105 TDataSet.OnPostError

Synopsis: Event triggered when a post operation fails.

Declaration: `Property OnPostError : TDataSetErrorEvent`

Visibility: public

Access: Read,Write

Description: `OnPostError` is triggered when the TDataset.Post (515) method fails to post the changes in the dataset buffer to the underlying database, because the database engine reported an error. The event handler can be used to indicate what the response to the failed post operation should be. To this end, it gets the exception object passed to it (parameter `E`), and it can examine this object to return an appropriate action in the `DataAction` parameter. The following responses are supported:

daFailThe operation should fail (an exception will be raised).

daAbortThe operation should be aborted (edits are undone, and an EAbort exception is raised).

daRetryRetry the operation.

For more information, see also the description of the TDatasetErrorEvent (443) event handler type.

See also: TDatasetErrorEvent (443), TDataset.Post (515), TDataset.OnDeleteError (533), TDataset.OnEditError (533)

16.24 TDataSetEnumerator

16.24.1 Description

`TDataSetEnumerator` is an enumerator for the records in a dataset. It returns the TDataset.Fields (524) instance. It navigates from the first till the last record in the dataset. The following is an example of how this can be used, in conjunction with the field enumerator:

```
var
  Rec : TFields;
  Fld : TField;

begin
  for Rec in MyDataset do
    for F in Rec do
      Writeln(F.Name,' : ',F.AsString);
```

Note that the current record pointer of the dataset is modified as the loop is traversed. If the current record is modified by other code while the loop is running, the result may become unpredictable. Similarly, if 2 enumerators are used simultaneously for the same dataset, the results are unpredictable.

See also: TDataset (496), TFields (584), TFieldsEnumerator (588)

16.24.2 Method overview

Page	Method	Description
535	Create	Create a new instance of the dataset enumerator.
535	MoveNext	Attempts to navigate to the next record.

16.24.3 Property overview

Page	Properties	Access	Description
536	Current	r	Current record.

16.24.4 TDataSetEnumerator.Create

Synopsis: Create a new instance of the dataset enumerator.

Declaration: `constructor Create(ADataSet: TDataSet)`

Visibility: public

Description: `Create` saves the dataset for later use, and puts the dataset on the firs record.

Errors: None.

See also: TDataset (496), TDataset.First (510), TFieldsEnumerator (588)

16.24.5 TDataSetEnumerator.MoveNext

Synopsis: Attempts to navigate to the next record.

Declaration: `function MoveNext : Boolean`

Visibility: public

Description: `MoveNext` attempts to navigate to the next record. It returns `True` if the attempt was successful, `False` if not (EOF is true).

See also: TDataset.Next (514), TDataset.EOF (520)

16.24.6 TDataSetEnumerator.Current

Synopsis: Current record.

Declaration: `Property Current : TFields`

Visibility: public

Access: Read

Description: `Current` always returns TDataset.Fields (524).

See also: TDataset.Fields (524)

16.25 TDataSource

16.25.1 Description

`TDatasource` is a mediating component: it handles communication between any DB-Aware component (often edit controls on a form) and a TDataset (496) instance. Any database aware component should never communicate with a dataset directly. Instead, it should communicate with a TDatasource (536) instance. The TDataset instance will communicate with the TDatasource instance, which will notify every component attached to it. Vice versa, any component that wishes to make changes to the dataset, will notify the TDatasource instance, which will then (if needed) notify the TDataset instance. The datasource can be disabled, in which case all communication between the dataset and the DB-AWare components is suspended until the datasource is again enabled.

See also: TDataset (496), TDatalink (491)

16.25.2 Method overview

Page	Method	Description
537	Create	Create a new instance of `TDatasource`.
537	Destroy	Remove a `TDatasource` instance from memory.
537	Edit	Put the dataset in edit mode, if needed.
537	IsLinkedTo	Check if a dataset is linked to a certain dataset.

16.25.3 Property overview

Page	Properties	Access	Description
538	AutoEdit	rw	Should the dataset be put in edit mode automatically.
538	DataSet	rw	Dataset this datasource is connected to.
539	Enabled	rw	Enable or disable sending of events.
539	OnDataChange	rw	Called whenever data changes in the current record.
539	OnStateChange	rw	Called whenever the state of the dataset changes.
540	OnUpdateData	rw	Called whenever the data in the dataset must be updated.
538	State	r	State of the dataset.

16.25.4 TDataSource.Create

Synopsis: Create a new instance of `TDatasource`.

Declaration: `constructor Create(AOwner: TComponent); Override`

Visibility: public

Description: `Create` initializes a new instance of `TDatasource`. It simply allocates some resources and then calls the inherited constructor.

See also: TDatasource.Destroy (537)

16.25.5 TDataSource.Destroy

Synopsis: Remove a `TDatasource` instance from memory.

Declaration: `destructor Destroy; Override`

Visibility: public

Description: `Destroy` notifies all TDataLink (491) instances connected to it that the dataset is no loner available, and then removes itself from the `TDatalink` instance. It then cleans up all resources and calls the inherited constructor.

See also: TDatasource.Create (537), TDatalink (491)

16.25.6 TDataSource.Edit

Synopsis: Put the dataset in edit mode, if needed.

Declaration: `procedure Edit`

Visibility: public

Description: `Edit` will check AutoEdit (538): if it is `True`, then it puts the Dataset (538) it is connected to in edit mode, if it was in browse mode. If `AutoEdit` is `False`, then nothing happens. Application or component code that deals with GUI development should always attempt to set a dataset in edit mode through this method instead of calling TDataset.Edit (507) directly.

Errors: An EDatabaseError (458) exception can occur if the dataset is read-only or fails to set itself in edit mode. (e.g. unidirectional datasets).

See also: TDatasource.AutoEdit (538), TDataset.Edit (507), TDataset.State (524)

16.25.7 TDataSource.IsLinkedTo

Synopsis: Check if a dataset is linked to a certain dataset.

Declaration: `function IsLinkedTo(ADataset: TDataSet) : Boolean`

Visibility: public

Description: `IsLinkedTo` checks if it is somehow linked to `ADataset`: it checks the Dataset (538) property, and returns `True` if it is the same. If not, it continues by checking any detail dataset fields that the dataset possesses (recursively). This function can be used to detect circular links in e.g. master-detail relationships.

See also: TDatasource.Dataset (538)

16.25.8 TDataSource.State

Synopsis: State of the dataset.

Declaration: `Property State : TDataSetState`

Visibility: public

Access: Read

Description: `State` contains the State (524) of the dataset it is connected to, or `dsInactive` if the dataset property is not set or the datasource is not enabled. Components connected to a dataset through a datasource property should always check `TDatasource.State` instead of checking TDataset.State (524) directly, to take into account the effect of the Enabled (539) property.

See also: TDataset.State (524), TDatasource.Enabled (539)

16.25.9 TDataSource.AutoEdit

Synopsis: Should the dataset be put in edit mode automatically.

Declaration: `Property AutoEdit : Boolean`

Visibility: published

Access: Read,Write

Description: `AutoEdit` can be set to `True` to prevent visual controls from putting the dataset in edit mode. Visual controls use the TDatasource.Edit (537) method to attempt to put the dataset in edit mode as soon as the user changes something. If `AutoEdit` is set to `False` then the `Edit` method does nothing. The effect is that the user must explicitly set the dataset in edit mode (by clicking some button or some other action) before the fields can be edited.

See also: TDatasource.Edit (537), TDataset.Edit (507)

16.25.10 TDataSource.DataSet

Synopsis: Dataset this datasource is connected to.

Declaration: `Property DataSet : TDataSet`

Visibility: published

Access: Read,Write

Description: `Dataset` must be set by the application programmer to the TDataset (496) instance for which this datasource is handling events. Setting it to `Nil` will disable all controls that are connected to this datasource instance. Once it is set and the datasource is enabled, the datasource will start sending data events to the controls or components connected to it.

See also: TDataset (496), TDatasource.Enabled (539)

16.25.11 TDataSource.Enabled

Synopsis: Enable or disable sending of events.

Declaration: `Property Enabled : Boolean`

Visibility: published

Access: Read,Write

Description: `Enabled` is by default set to `True`: the datasource instance communicates events from the dataset to components connected to the datasource, and vice versa: components can interact with the dataset. If the `Enabled` property is set to `False` then no events are communicated to connected components: it is as if the dataset property was set to `Nil`. Reversely, the components cannot interact with the dataset if the `Enabled` property is set to `False`.

See also: TDataset (496), TDatasource.Dataset (538), TDatasource.AutoEdit (538)

16.25.12 TDataSource.OnStateChange

Synopsis: Called whenever the state of the dataset changes.

Declaration: `Property OnStateChange : TNotifyEvent`

Visibility: published

Access: Read,Write

Description: `OnStateChange` is called whenever the TDataset.State (524) property changes, and the datasource is enabled. It can be used in application code to react to state changes: enabling or disabling non-DB-Aware controls, setting empty values etc.

See also: TDatasource.OnUpdateData (540), TDatasource.OnStateChange (539), TDataset.State (524), TDatasource.Enabled (539)

16.25.13 TDataSource.OnDataChange

Synopsis: Called whenever data changes in the current record.

Declaration: `Property OnDataChange : TDataChangeEvent`

Visibility: published

Access: Read,Write

Description: `OnDatachange` is called whenever a field value changes: if the `Field` parameter is set, a single field value changed. If the `Field` parameter is `Nil`, then the whole record changed: when the dataset is opened, when the user scrolls to a new record. This event handler can be set to react to data changes: to update the contents of non-DB-aware controls for instance. The event is not called when the datasource is not enabled.

See also: TDatasource.OnUpdateData (540), TDatasource.OnStateChange (539), TDataset.AfterScroll (531), TField.OnChange (574), TDatasource.Enabled (539)

16.25.14 TDataSource.OnUpdateData

Synopsis: Called whenever the data in the dataset must be updated.

Declaration: `Property OnUpdateData : TNotifyEvent`

Visibility: published

Access: Read,Write

Description: `OnUpdateData` is called whenever the dataset needs the latest data from the controls: usually just before a TDataset.Post (515) operation. It can be used to copy data from non-db-aware controls to the dataset just before the dataset is posting the changes to the underlying database.

See also: TDatasource.OnDataChange (539), TDatasource.OnStateChange (539), TDataset.Post (515)

16.26 TDateField

16.26.1 Description

`TDateField` is the class used when a dataset must manage data of type date. (TField.DataType (562) equals `ftDate`). It initializes some of the properties of the TField (549) class to be able to work with date fields.

It should never be necessary to create an instance of `TDateField` manually, a field of this class will be instantiated automatically for each date field when a dataset is opened.

See also: TDataset (496), TField (549), TDateTimeField (540), TTimeField (643)

16.26.2 Method overview

Page	Method	Description
540	Create	Create a new instance of a `TDateField` class.

16.26.3 TDateField.Create

Synopsis: Create a new instance of a `TDateField` class.

Declaration: `constructor Create(AOwner: TComponent); Override`

Visibility: public

Description: `Create` initializes a new instance of the `TDateField` class. It calls the inherited destructor, and then sets some TField (549) properties to configure the instance for working with date values.

See also: TField (549)

16.27 TDateTimeField

16.27.1 Description

`TDateTimeField` is the class used when a dataset must manage data of type datetime. (TField.DataType (562) equals `ftDateTime`). It also serves as base class for the TDateField (540) or TTimeField (643) classes. It overrides some of the properties and methods of the TField (549) class to be able to work with date/time fields.

It should never be necessary to create an instance of `TDateTimeField` manually, a field of this class will be instantiated automatically for each datetime field when a dataset is opened.

See also: TDataset (496), TField (549), TDateField (540), TTimeField (643)

16.27.2 Method overview

Page	Method	Description
541	Create	Create a new instance of a `TDateTimeField` class.

16.27.3 Property overview

Page	Properties	Access	Description
541	DisplayFormat	rw	Formatting string for textual representation of the field.
542	EditMask		Specify an edit mask for an edit control.
541	Value	rw	Contents of the field as a TDateTime value.

16.27.4 TDateTimeField.Create

Synopsis: Create a new instance of a `TDateTimeField` class.

Declaration: `constructor Create(AOwner: TComponent);  Override`

Visibility: public

Description: `Create` initializes a new instance of the `TDateTimeField` class. It calls the inherited destructor, and then sets some TField (549) properties to configure the instance for working with date/time values.

See also: TField (549)

16.27.5 TDateTimeField.Value

Synopsis: Contents of the field as a TDateTime value.

Declaration: `Property Value : TDateTime`

Visibility: public

Access: Read,Write

Description: `Value` is redefined from TField.Value (566) by `TDateTimeField` as a `TDateTime` value. It returns the same value as the TField.AsDateTime (557) property.

See also: TField.AsDateTime (557), TField.Value (566)

16.27.6 TDateTimeField.DisplayFormat

Synopsis: Formatting string for textual representation of the field.

Declaration: `Property DisplayFormat : string`

Visibility: published

Access: Read,Write

Description: `DisplayFormat` can be set to a formatting string that will then be used by the TField.DisplayText (563) property to format the value with the DateTimeToString (**??**)function.

See also: DateTimeToString (**??**), FormatDateTime (**??**), TField.DisplayText (563)

16.27.7 TDateTimeField.EditMask

Synopsis: Specify an edit mask for an edit control.

Declaration: `Property EditMask :`

Visibility: published

Access:

Description: `EditMask` can be used to specify an edit mask for controls that allow to edit this field. It has no effect on the field value, and serves only to ensure that the user can enter only correct data for this field.

`TDateTimeField` just changes the visibility of the `EditMark` property, it is introduced in `TField`.

For more information on valid edit masks, see the documentation of the GUI controls.

See also: TField.EditMask (563)

16.28 TDBDataset

16.28.1 Description

`TDBDataset` is a `TDataset` descendent which introduces the concept of a database: a central component (TDatabase (487)) which represents a connection to a database. This central component is exposed in the TDBDataset.Database (543) property. When the database is no longer connected, or is no longer in memory, all `TDBDataset` instances connected to it are disabled.

`TDBDataset` also introduces the notion of a transaction, exposed in the Transaction (543) property.

`TDBDataset` is an abstract class, it should never be used directly.

Dataset component writers should descend their component from `TDBDataset` if they wish to introduce a central database connection component. The database connection logic will be handled automatically by `TDBDataset`.

See also: TDatabase (487), TDBTransaction (543)

16.28.2 Method overview

Page	Method	Description
543	destroy	Remove the `TDBDataset` instance from memory.

16.28.3 Property overview

Page	Properties	Access	Description
543	DataBase	rw	Database this dataset is connected to.
543	Transaction	rw	Transaction in which this dataset is running.

16.28.4 TDBDataset.destroy

Synopsis: Remove the `TDBDataset` instance from memory.

Declaration: `destructor destroy;  Override`

Visibility: public

Description: `Destroy` will disconnect the `TDBDataset` from its Database (543) and Transaction (543). After this it calls the inherited destructor.

See also: TDBDataset.Database (543), TDatabase (487)

16.28.5 TDBDataset.DataBase

Synopsis: Database this dataset is connected to.

Declaration: `Property DataBase : TDataBase`

Visibility: public

Access: Read,Write

Description: `Database` should be set to the TDatabase (487) instance this dataset is connected to. It can only be set when the dataset is closed.

Descendent classes should check in the property setter whether the database instance is of the correct class.

Errors: If the property is set when the dataset is active, an EDatabaseError (458) exception will be raised.

See also: TDatabase (487), TDBDataset.Transaction (543)

16.28.6 TDBDataset.Transaction

Synopsis: Transaction in which this dataset is running.

Declaration: `Property Transaction : TDBTransaction`

Visibility: public

Access: Read,Write

Description: `Transaction` points to a TDBTransaction (543) component that represents the transaction this dataset is active in. This property should only be used for databases that support transactions.

The property can only be set when the dataset is disabled.

See also: TDBTransaction (543), TDBDataset.Database (543)

16.29 TDBTransaction

16.29.1 Description

`TDBTransaction` encapsulates a SQL transaction. It is an abstract class, and should be used by component creators that wish to encapsulate transactions in a class. The `TDBTransaction` class offers functionality to refer to a TDatabase (487) instance, and to keep track of `TDataset` instances which are connected to the transaction.

See also: TDatabase (487), TDataset (496)

16.29.2 Method overview

Page	Method	Description
544	CloseDataSets	Close all connected datasets.
544	Create	Transaction property.
544	Destroy	Remove a `TDBTransaction` instance from memory.

16.29.3 Property overview

Page	Properties	Access	Description
545	Active	rw	Is the transaction active or not.
545	DataBase	rw	Database this transaction is connected to.

16.29.4 TDBTransaction.Create

Synopsis: Transaction property.

Declaration:
```
constructor Create(AOwner: TComponent);  Override
```

Visibility: public

Description: `Create` initializes a new `TDBTransaction` instance. It sets up the necessary resources, after having called the inherited constructor.

See also: TDBTransaction.Destroy (544)

16.29.5 TDBTransaction.Destroy

Synopsis: Remove a `TDBTransaction` instance from memory.

Declaration:
```
destructor Destroy;  Override
```

Visibility: public

Description: `Destroy` first disconnects all connected TDBDataset (542) instances and then cleans up the resources allocated in the Create (544) constructor. After that it calls the inherited destructor.

See also: TDBTransaction.Create (544)

16.29.6 TDBTransaction.CloseDataSets

Synopsis: Close all connected datasets.

Declaration:
```
procedure CloseDataSets
procedure CloseDataSets(InCommit: Boolean)
```

Visibility: public

Description: `CloseDatasets` closes all connected datasets (All TDBDataset (542) instances whose Transaction (543) property points to this `TDBTransaction` instance).

See also: TDBDataset (542), TDBDataset.Transaction (543)

16.29.7 TDBTransaction.DataBase

Synopsis: Database this transaction is connected to.

Declaration: `Property DataBase : TDataBase`

Visibility: public

Access: Read,Write

Description: `Database` points to the database that this transaction is part of. This property can be set only when the transaction is not active.

Errors: Setting this property to a new value when the transaction is active will result in an EDatabaseError (458) exception.

See also: TDBTransaction.Active (545), TDatabase (487)

16.29.8 TDBTransaction.Active

Synopsis: Is the transaction active or not.

Declaration: `Property Active : Boolean`

Visibility: published

Access: Read,Write

Description: `Active` is `True` if a transaction was started using TDBTransaction.StartTransaction (543). Reversely, setting `Active` to `True` will call `StartTransaction`, setting it to `False` will call TDBTransaction.EndTransaction (543).

See also: TDBTransaction.StartTransaction (543), TDBTransaction.EndTransaction (543)

16.30 TDefCollection

16.30.1 Description

`TDefCollection` is a parent class for the TFieldDefs (581) and TIndexDefs (599) collections: It holds a set of named definitions on behalf of a TDataset (496) component. To this end, it introduces a dataset (547) property, and a mechanism to notify the dataset of any updates in the collection. It is supposed to hold items of class TNamedItem (614), so the TDefCollection.Find (546) method can find items by named.

16.30.2 Method overview

Page	Method	Description
546	create	Instantiate a new `TDefCollection` instance.
546	Find	Find an item by name.
546	GetItemNames	Return a list of all names in the collection.
546	IndexOf	Find location of item by name.

16.30.3 Property overview

Page	Properties	Access	Description
547	Dataset	r	Dataset this collection manages definitions for.
547	Updated	rw	Has one of the items been changed.

16.30.4 TDefCollection.create

Synopsis: Instantiate a new `TDefCollection` instance.

Declaration:
```
constructor create(ADataset: TDataSet; AOwner: TPersistent;
                   AClass: TCollectionItemClass)
```

Visibility: public

Description: `Create` saves the `ADataset` and `AOwner` components in local variables for later reference, and then calls the inherited `Create` with `AClass` as a parameter. `AClass` should at least be of type `TNamedItem`. `ADataset` is the dataset on whose behalf the collection is managed. `AOwner` is the owner of the collection, normally this is the form or datamodule on which the dataset is dropped.

See also: TDataset (496), TNamedItem (614)

16.30.5 TDefCollection.Find

Synopsis: Find an item by name.

Declaration:
```
function Find(const AName: string) : TNamedItem
```

Visibility: public

Description: `Find` searches for an item in the collection with name `AName` and returns the item if it is found. If no item with the requested name is found, `Nil` is returned. The search is performed case-insensitive.

Errors: If no item with matching name is found, `Nil` is returned.

See also: TNamedItem.Name (614), TDefCollection.IndexOf (546)

16.30.6 TDefCollection.GetItemNames

Synopsis: Return a list of all names in the collection.

Declaration:
```
procedure GetItemNames(List: TStrings)
```

Visibility: public

Description: `GetItemNames` fills `List` with the names of all items in the collection. It clears the list first.

Errors: If `List` is not a valid `TStrings` instance, an exception will occur.

See also: TNamedItem.Name (614)

16.30.7 TDefCollection.IndexOf

Synopsis: Find location of item by name.

Declaration:
```
function IndexOf(const AName: string) : LongInt
```

Visibility: public

Description: `IndexOf` searches in the collection for an item whose `Name` property matches `AName` and returns the index of the item if it finds one. If no item is found, -1 is returned. The search is performed case-insensitive.

See also: TDefCollection.Find (546), TNamedItem.Name (614)

16.30.8 TDefCollection.Dataset

Synopsis: Dataset this collection manages definitions for.

Declaration: `Property Dataset : TDataSet`

Visibility: public

Access: Read

Description: `Dataset` is the dataset this collection manages definitions for. It must be supplied when the collection is created and cannot change during the lifetime of the collection.

16.30.9 TDefCollection.Updated

Synopsis: Has one of the items been changed.

Declaration: `Property Updated : Boolean`

Visibility: public

Access: Read,Write

Description: `Changed` indicates whether the collection has changed: an item was added or removed, or one of the properties of the items was changed.

16.31 TDetailDataLink

16.31.1 Description

`TDetailDataLink` handles the communication between a detail dataset and the master datasource in a master-detail relationship between datasets. It should never be used in an application, and should only be used by component writers that wish to provide master-detail functionality for TDataset descendents.

See also: TDataset (496), TDatasource (536)

16.31.2 Property overview

Page	Properties	Access	Description
547	DetailDataSet	r	Detail dataset in Master-detail relation.

16.31.3 TDetailDataLink.DetailDataSet

Synopsis: Detail dataset in Master-detail relation.

Declaration: `Property DetailDataSet : TDataSet`

Visibility: public

Access: Read

Description: `DetailDataset` is the detail dataset in a master-detail relationship between 2 datasets. `DetailDataset` is always `Nil` in `TDetailDatalink` and is only filled in in descendent classes like TMasterDatalink (609). The master dataset is available through the regular TDataLink.DataSource (495) property.

See also: TDataset (496), TMasterDatalink (609), TDataLink.DataSource (495)

16.32 TExtendedField

16.32.1 Method overview

Page	Method	Description
548	CheckRange	
548	Create	

16.32.2 Property overview

Page	Properties	Access	Description
548	Currency	rw	
548	MaxValue	rw	
549	MinValue	rw	
549	Precision	rw	
548	Value	rw	

16.32.3 TExtendedField.Create

Declaration: `constructor Create(AOwner: TComponent); Override`

Visibility: public

16.32.4 TExtendedField.CheckRange

Declaration: `function CheckRange(AValue: Extended) : Boolean`

Visibility: public

16.32.5 TExtendedField.Value

Declaration: `Property Value : Extended`

Visibility: public

Access: Read,Write

16.32.6 TExtendedField.Currency

Declaration: `Property Currency : Boolean`

Visibility: published

Access: Read,Write

16.32.7 TExtendedField.MaxValue

Declaration: `Property MaxValue : Extended`

Visibility: published

Access: Read,Write

16.32.8 TExtendedField.MinValue

Declaration: `Property MinValue : Extended`

Visibility: published

Access: Read,Write

16.32.9 TExtendedField.Precision

Declaration: `Property Precision : LongInt`

Visibility: published

Access: Read,Write

16.33 TField

16.33.1 Description

`TField` is an abstract class that defines access methods for a field in a record, controlled by a TDataset (496) instance. It provides methods and properties to access the contents of the field in the current record. Reading one of the AsXXX properties of `TField` will access the field contents and return the contents as the desired type. Writing one of the AsXXX properties will write a value to the buffer represented by the `TField` instance.

`TField` is an abstract class, meaning that it should never be created directly. `TDataset` instances always create one of the descendent classes of TField, depending on the type of the underlying data.

See also: TDataset (496), TFieldDef (575), TFields (584)

16.33.2 Method overview

Page	Method	Description
552	Assign	Copy properties from one TField instance to another.
552	AssignValue	Assign value of a variant record to the field.
553	Clear	Clear the field contents.
552	Create	Create a new `TField` instance.
552	Destroy	Destroy the `TField` instance.
553	FocusControl	Set focus to the first control connected to this field.
553	GetData	Get the data from this field.
554	IsBlob	Is the field a BLOB field (untyped data of indeterminate size).
554	IsValidChar	Check whether a character is valid input for the field.
554	RefreshLookupList	Refresh the lookup list.
554	SetData	Save the field data.
555	SetFieldType	Set the field data type.
555	Validate	Validate the data buffer.

16.33.3 Property overview

Page	Properties	Access	Description
567	Alignment	rw	Alignment for this field.
559	AsAnsiString	rw	Return field contents as an ANSI string.
555	AsBCD	rw	Access the field's contents as a BCD (Binary coded Decimal).
556	AsBoolean	rw	Access the field's contents as a Boolean value.
556	AsBytes	rw	Retrieve the contents of the field as an array of bytes.
556	AsCurrency	rw	Access the field's contents as a Currency value.
557	AsDateTime	rw	Access the field's contents as a TDateTime value.
557	AsExtended	rw	
557	AsFloat	rw	Access the field's contents as a floating-point (Double) value.
558	AsInteger	rw	Access the field's contents as a 32-bit signed integer (longint) value.
558	AsLargeInt	rw	Access the field's contents as a 64-bit signed integer (longint) value.
557	AsLongint	rw	Access the field's contents as a 32-bit signed integer (longint) value.
558	AsLongWord	rw	Access field contents as 32-bit unsigned integer (longword/cardinal).
559	AsSingle	rw	
559	AsString	rw	Access the field's contents as an AnsiString value.
560	AsUnicodeString	rw	Field contents as a UnicodeString.
560	AsUTF8String	rw	Field contents as a UTF8 String.
561	AsVariant	rw	Access the field's contents as a Variant value.
560	AsWideString	rw	Access the field's contents as a WideString value.
561	AttributeSet	rw	Not used: dictionary information.
561	Calculated	rw	Is the field a calculated field ?
561	CanModify	r	Can the field's contents be modified.
568	ConstraintErrorMessage	rw	Message to display if the `CustomConstraint` constraint is violated.
562	CurValue	r	Current value of the field.
568	CustomConstraint	rw	Custom constraint for the field's value.
562	DataSet	rw	Dataset this field belongs to.
562	DataSize	r	Size of the field's data.
562	DataType	r	The data type of the field.
568	DefaultExpression	rw	Default value for the field.
569	DisplayLabel	rws	Name of the field for display purposes.
563	DisplayName	r	User-readable fieldname.
563	DisplayText	r	Formatted field value.
569	DisplayWidth	rws	Width of the field in characters.
563	EditMask	rw	Specify an edit mask for an edit control.
564	EditMaskPtr	r	Alias for EditMask.
567	FieldDef	r	Fielddef associated with this field.
569	FieldKind	rw	The kind of field.
569	FieldName	rw	Name of the field.
564	FieldNo	r	Number of the field in the record.
570	HasConstraints	r	Does the field have any constraints defined.
570	ImportedConstraint	rw	Constraint for the field value on the level of the underlying database.
570	Index	rw	Index of the field in the list of fields.
564	IsIndexField	r	Is the field an indexed field ?
564	IsNull	r	Is the field empty.
571	KeyFields	rw	Key fields to use when looking up a field value.
572	Lookup	rws	Is the field a lookup field.
571	LookupCache	rw	Should lookup values be cached.
571	LookupDataSet	rw	Dataset with lookup values.
571	LookupKeyFields	rw	Names of fields on which to perform a locate.
567	LookupList	r	List of lookup values.
572	LookupResultField	rw	Name of field to use as lookup value.

16.33.4 TField.Create

Synopsis: Create a new `TField` instance.

Declaration: `constructor Create(AOwner: TComponent);  Override`

Visibility: public

Description: `Create` creates a new `TField` instance and sets up initial values for the fields. `TField` is a component, and `AOwner` will be used as the owner of the `TField` instance. This usually will be the form or datamodule on which the dataset was placed. There should normally be no need for a programmer to create a Tfield instance manually. The TDataset.Open (515) method will create the necessary `TField` instances, if none had been created in the designer.

See also: TDataset.Open (515)

16.33.5 TField.Destroy

Synopsis: Destroy the `TField` instance.

Declaration: `destructor Destroy;  Override`

Visibility: public

Description: `Destroy` cleans up any structures set up by the field instance, and then calls the inherited destructor. There should be no need to call this method under normal circumstances: the dataset instance will free any TField instances it has created when the dataset was opened.

See also: TDataset.Close (505)

16.33.6 TField.Assign

Synopsis: Copy properties from one TField instance to another.

Declaration: `procedure Assign(Source: TPersistent);  Override`

Visibility: public

Description: `Assign` is overridden by `TField` to copy the field value (not the field properties) from `Source` if it exists. If `Source` is `Nil` then the value of the field is cleared.

Errors: If `Source` is not a `TField` instance, then an exception will be raised.

See also: TField.Value (566)

16.33.7 TField.AssignValue

Synopsis: Assign value of a variant record to the field.

Declaration: `procedure AssignValue(const AValue: TVarRec)`

Visibility: public

Description: `AssignValue` assigns the value of a "array of const" record `AValue` (of type `TVarRec`) to the field's value. If the record contains a TPersistent instance, it will be used as argument for the `Assign` to the field.

The dataset must be in edit mode to execute this method.

Errors: If the `AValue` contains an unsupported value (such as a non-nil pointer) then an exception will be raised. If the dataset is not in one of the edit modes, then executing this method will raise an EDatabaseError (458) exception.

See also: TField.Assign (552), TField.Value (566)

16.33.8 TField.Clear

Synopsis: Clear the field contents.

Declaration:
```
procedure Clear;  Virtual
```

Visibility: public

Description: `Clear` clears the contents of the field. After calling this method the value of the field is `Null` and IsNull (564) returns `True`.

The dataset must be in edit mode to execute this method.

Errors: If the dataset is not in one of the edit modes, then executing this method will raise an EDatabaseError (458) exception.

See also: TField.IsNull (564), TField.Value (566)

16.33.9 TField.FocusControl

Synopsis: Set focus to the first control connected to this field.

Declaration:
```
procedure FocusControl
```

Visibility: public

Description: `FocusControl` will set focus to the first control that is connected to this field.

Errors: If the control cannot receive focus, then this method will raise an exception.

See also: TDataset.EnableControls (508), TDataset.DisableControls (507)

16.33.10 TField.GetData

Synopsis: Get the data from this field.

Declaration:
```
function GetData(Buffer: Pointer) : Boolean;  Overload
function GetData(Buffer: Pointer; NativeFormat: Boolean) : Boolean
                ;  Overload
```

Visibility: public

Description: `GetData` is used internally by `TField` to fetch the value of the data of this field into the data buffer pointed to by `Buffer`. If it returns `False` if the field has no value (i.e. is Null). If the `NativeFormat` parameter is true, then date/time formats should use the `TDateTime` format. It should not be necessary to use this method, instead use the various 'AsXXX' methods to access the data.

Errors: No validity checks are performed on `Buffer`: it should point to a valid memory area, and should be large enough to contain the value of the field. Failure to provide a buffer that matches these criteria will result in an exception.

See also: TField.IsNull (564), TField.SetData (554), TField.Value (566)

16.33.11 TField.IsBlob

Synopsis: Is the field a BLOB field (untyped data of indeterminate size).

Declaration:
```
class function IsBlob : Boolean;  Virtual
```

Visibility: public

Description: `IsBlob` returns `True` if the field is one of the blob field types. The `TField` implementation returns false. Only one of the blob-type field classes override this function and let it return `True`.

Errors: None.

See also: TBlobField.IsBlob (472)

16.33.12 TField.IsValidChar

Synopsis: Check whether a character is valid input for the field.

Declaration:
```
function IsValidChar(InputChar: Char) : Boolean;  Virtual
```

Visibility: public

Description: `IsValidChar` checks whether `InputChar` is a valid characters for the current field. It does this by checking whether `InputChar` is in the set of characters specified by the TField.ValidChars (566) property. The `ValidChars` property will be initialized to a correct set of characters by descendent classes. For instance, a numerical field will only accept numerical characters and the sign and decimal separator characters.

Descendent classes can override this method to provide custom checks. The `ValidChars` property can be set to restrict the list of valid characters to a subset of what would normally be available.

See also: TField.ValidChars (566)

16.33.13 TField.RefreshLookupList

Synopsis: Refresh the lookup list.

Declaration:
```
procedure RefreshLookupList
```

Visibility: public

Description: `RefreshLookupList` fills the lookup list for a lookup fields with all key, value pairs found in the lookup dataset. It will open the lookup dataset if needed. The lookup list is only used if the TField.LookupCache (571) property is set to `True`.

Errors: If the values of the various lookup properties is not correct or the lookup dataset cannot be opened, then an exception will be raised.

See also: LookupDataset (571), LookupKeyFields (571), LookupResultField (572)

16.33.14 TField.SetData

Synopsis: Save the field data.

Declaration:
```
procedure SetData(Buffer: Pointer);  Overload
procedure SetData(Buffer: Pointer; NativeFormat: Boolean);  Overload
```

Visibility: public

Description: `SetData` saves the value of the field data in `Buffer` to the dataset internal buffer. The `Buffer` pointer should point to a memory buffer containing the data for the field in the correct format. If the `NativeFormat` parameter is true, then date/time formats should use the `TDateTime` format.

There should normally not be any need to call `SetData` directly: it is called by the various setter methods of the AsXXX properties of `TField`.

Errors: No validity checks are performed on `Buffer`: it should point to a valid memory area, and should be large enough to contain the value of the field. Failure to provide a buffer that matches these criteria will result in an exception.

See also: TField.GetData (553), TField.Value (566)

16.33.15 TField.SetFieldType

Synopsis: Set the field data type.

Declaration: `procedure SetFieldType(AValue: TFieldType); Virtual`

Visibility: public

Description: `SetFieldType` does nothing, but it can be overridden by descendent classes to provide special handling when the field type is set.

See also: TField.DataType (562)

16.33.16 TField.Validate

Synopsis: Validate the data buffer.

Declaration: `procedure Validate(Buffer: Pointer)`

Visibility: public

Description: `Validate` is called by `SetData` prior to writing the data from `Buffer` to the dataset buffer. It will call the TField.OnValidate (575) event handler, if one is set, to allow the application programmer to program additional checks.

See also: TField.SetData (554), TField.OnValidate (575)

16.33.17 TField.AsBCD

Synopsis: Access the field's contents as a BCD (Binary coded Decimal).

Declaration: `Property AsBCD : TBCD`

Visibility: public

Access: Read,Write

Description: `AsBCD` can be used to read or write the contents of the field as a BCD value (Binary Coded Decimal). If the native type of the field is not BCD, then an attempt will be made to convert the field value from the native format to a BCD value when reading the field's content. Likewise, when writing the property, the value will be converted to the native type of the field (if the value allows it). Therefor, when reading or writing a field value for a field whose native data type is not a BCD value, an exception may be raised.

See also: TField.AsCurrency (556), TField.Value (566)

16.33.18 TField.AsBoolean

Synopsis: Access the field's contents as a Boolean value.

Declaration: `Property AsBoolean : Boolean`

Visibility: public

Access: Read,Write

Description: `AsBoolean` can be used to read or write the contents of the field as a boolean value. If the native type of the field is not Boolean, then an attempt will be made to convert the field value from the native format to a boolean value when reading the field's content. Likewise, when writing the property, the value will be converted to the native type of the field (if the value allows it). Therefor, when reading or writing a field value for a field whose native data type is not a Boolean value (for instance a string value), an exception may be raised.

See also: TField.Value (566), TField.AsInteger (558)

16.33.19 TField.AsBytes

Synopsis: Retrieve the contents of the field as an array of bytes.

Declaration: `Property AsBytes : TBytes`

Visibility: public

Access: Read,Write

Description: `AsBytes` returns the contents of the field as an array of bytes. For blob data this is the actual blob content.

See also: TBlobField (471)

16.33.20 TField.AsCurrency

Synopsis: Access the field's contents as a Currency value.

Declaration: `Property AsCurrency : Currency`

Visibility: public

Access: Read,Write

Description: `AsBoolean` can be used to read or write the contents of the field as a currency value. If the native type of the field is not Boolean, then an attempt will be made to convert the field value from the native format to a currency value when reading the field's content. Likewise, when writing the property, the value will be converted to the native type of the field (if the value allows it). Therefor, when reading or writing a field value for a field whose native data type is not a currency-compatible value (dates or string values), an exception may be raised.

See also: TField.Value (566), TField.AsFloat (557)

16.33.21 TField.AsDateTime

Synopsis: Access the field's contents as a TDateTime value.

Declaration: `Property AsDateTime : TDateTime`

Visibility: public

Access: Read,Write

Description: `AsDateTime` can be used to read or write the contents of the field as a TDateTime value (for both date and time values). If the native type of the field is not a date or time value, then an attempt will be made to convert the field value from the native format to a TDateTime value when reading the field's content. Likewise, when writing the property, the value will be converted to the native type of the field (if the value allows it). Therefor, when reading or writing a field value for a field whose native data type is not a TDateTime-compatible value (dates or string values), an exception may be raised.

See also: TField.Value (566), TField.AsString (559)

16.33.22 TField.AsExtended

Declaration: `Property AsExtended : Extended`

Visibility: public

Access: Read,Write

16.33.23 TField.AsFloat

Synopsis: Access the field's contents as a floating-point (Double) value.

Declaration: `Property AsFloat : Double`

Visibility: public

Access: Read,Write

Description: `AsFloat` can be used to read or write the contents of the field as a floating-point value (of type double, i.e. with double precision). If the native type of the field is not a floating-point value, then an attempt will be made to convert the field value from the native format to a floating-point value when reading the field's content. Likewise, when writing the property, the value will be converted to the native type of the field (if the value allows it). Therefor, when reading or writing a field value for a field whose native data type is not a floating-point-compatible value (string values for instance), an exception may be raised.

See also: TField.Value (566), TField.AsString (559), TField.AsCurrency (556)

16.33.24 TField.AsLongint

Synopsis: Access the field's contents as a 32-bit signed integer (longint) value.

Declaration: `Property AsLongint : LongInt`

Visibility: public

Access: Read,Write

Description: `AsLongint` can be used to read or write the contents of the field as a 32-bit signed integer value (of type longint). If the native type of the field is not a longint value, then an attempt will be made to convert the field value from the native format to a longint value when reading the field's content. Likewise, when writing the property, the value will be converted to the native type of the field (if the value allows it). Therefor, when reading or writing a field value for a field whose native data type is not a 32-bit signed integer-compatible value (string values for instance), an exception may be raised.

This is an alias for the TField.AsInteger (558).

See also: TField.Value (566), TField.AsString (559), TField.AsInteger (558)

16.33.25 TField.AsLongWord

Synopsis: Access field contents as 32-bit unsigned integer (longword/cardinal).

Declaration: `Property AsLongWord : LongWord`

Visibility: public

Access: Read,Write

Description: `AsInteger` can be used to read or write the contents of the field as a 32-bit signed integer value (of type Integer). If the native type of the field is not an integer value, then an attempt will be made to convert the field value from the native format to a integer value when reading the field's content. Likewise, when writing the property, the value will be converted to the native type of the field (if the value allows it). Therefor, when reading or writing a field value for a field whose native data type is not a 32-bit unsigned integer-compatible value (string values for instance), an exception may be raised.

See also: TField.Value (566), TField.AsInteger (558)

16.33.26 TField.AsLargeInt

Synopsis: Access the field's contents as a 64-bit signed integer (longint) value.

Declaration: `Property AsLargeInt : LargeInt`

Visibility: public

Access: Read,Write

Description: `AsLargeInt` can be used to read or write the contents of the field as a 64-bit signed integer value (of type Int64). If the native type of the field is not an Int64 value, then an attempt will be made to convert the field value from the native format to an Int64 value when reading the field's content. Likewise, when writing the property, the value will be converted to the native type of the field (if the value allows it). Therefor, when reading or writing a field value for a field whose native data type is not a 64-bit signed integer-compatible value (string values for instance), an exception may be raised.

See also: TField.Value (566), TField.AsString (559), TField.AsInteger (558)

16.33.27 TField.AsInteger

Synopsis: Access the field's contents as a 32-bit signed integer (longint) value.

Declaration: `Property AsInteger : LongInt`

Visibility: public

Access: Read,Write

Description: `AsInteger` can be used to read or write the contents of the field as a 32-bit signed integer value (of type Integer). If the native type of the field is not an integer value, then an attempt will be made to convert the field value from the native format to a integer value when reading the field's content. Likewise, when writing the property, the value will be converted to the native type of the field (if the value allows it). Therefor, when reading or writing a field value for a field whose native data type is not a 32-bit signed integer-compatible value (string values for instance), an exception may be raised.

See also: TField.Value (566), TField.AsString (559), TField.AsLongint (557), TField.AsInt64 (549)

16.33.28 TField.AsSingle

Declaration: `Property AsSingle : Single`

Visibility: public

Access: Read,Write

16.33.29 TField.AsString

Synopsis: Access the field's contents as an AnsiString value.

Declaration: `Property AsString : string`

Visibility: public

Access: Read,Write

Description: `AsString` can be used to read or write the contents of the field as an AnsiString value. If the native type of the field is not an ansistring value, then an attempt will be made to convert the field value from the native format to a ansistring value when reading the field's content. Likewise, when writing the property, the value will be converted to the native type of the field (if the value allows it). Therefor, when reading or writing a field value for a field whose native data type is not an ansistring-compatible value, an exception may be raised.

See also: TField.Value (566), TField.AsWideString (560)

16.33.30 TField.AsAnsiString

Synopsis: Return field contents as an ANSI string.

Declaration: `Property AsAnsiString : AnsiString`

Visibility: public

Access: Read,Write

Description: `AsAnsiString` returns the field data as an ANSI string (single byte character string). Note that if the field contains unicode data, some characters may get lost when reading.

See also: TField.AsString (559), TField.AsUnicodeString (560), TField.AsUTF8String (560), TField.CodePage (549)

16.33.31 TField.AsUnicodeString

Synopsis: Field contents as a UnicodeString.

Declaration: `Property AsUnicodeString : UnicodeString`

Visibility: public

Access: Read,Write

Description: `AsUnicodeString` returns the field data as a Unicode string (double byte character string). If the field contains an AnsiString, the data will be converted to unicode according to the CodePage (549) when reading, and when writing the written data will be converted to single-byte string. Note that if the field is an ansistring field, some characters may get lost when writing.

See also: TField.AsString (559), TField.AsAnsiString (559), TField.AsUTF8String (560), TField.CodePage (549)

16.33.32 TField.AsUTF8String

Synopsis: Field contents as a UTF8 String.

Declaration: `Property AsUTF8String : UTF8String`

Visibility: public

Access: Read,Write

Description: `AsUTF8String` returns the field data as a UTF8-Encoded string (single byte character string). If the field contains an AnsiString, the data will be converted to unicode according to the CodePage (549). If the field contains a unicode string, the string is UTF-8 encoded. When writing the written data will be converted to single-byte string. Note that if the field is an ansistring field, some characters may get lost when writing.

See also: TField.AsString (559), TField.AsUnicodeString (560), TField.AsAnsi8String (549), TField.CodePage (549)

16.33.33 TField.AsWideString

Synopsis: Access the field's contents as a WideString value.

Declaration: `Property AsWideString : WideString`

Visibility: public

Access: Read,Write

Description: `AsWideString` can be used to read or write the contents of the field as a WideString value. If the native type of the field is not a widestring value, then an attempt will be made to convert the field value from the native format to a widestring value when reading the field's content. Likewise, when writing the property, the value will be converted to the native type of the field (if the value allows it). Therefor, when reading or writing a field value for a field whose native data type is not a widestring-compatible value, an exception may be raised.

See also: TField.Value (566), TField.Astring (549)

16.33.34 TField.AsVariant

Synopsis: Access the field's contents as a Variant value.

Declaration: `Property AsVariant : variant`

Visibility: public

Access: Read,Write

Description: `AsVariant` can be used to read or write the contents of the field as a Variant value. If the native type of the field is not a Variant value, then an attempt will be made to convert the field value from the native format to a variant value when reading the field's content. Likewise, when writing the property, the value will be converted to the native type of the field (if the value allows it). Therefor, when reading or writing a field value for a field whose native data type is not a variant-compatible value, an exception may be raised.

See also: TField.Value (566), TField.Astring (549)

16.33.35 TField.AttributeSet

Synopsis: Not used: dictionary information.

Declaration: `Property AttributeSet : string`

Visibility: public

Access: Read,Write

Description: `AttributeSet` was used in older Delphi versions to store data dictionary information for use in data-aware controls at design time. Not used in FreePascal (or newer Delphi versions); kept for Delphi compatibility.

16.33.36 TField.Calculated

Synopsis: Is the field a calculated field ?

Declaration: `Property Calculated : Boolean`

Visibility: public

Access: Read,Write

Description: `Calculated` is `True` if the FieldKind (569) is `fkCalculated`. Setting the property will result in `FieldKind` being set to `fkCalculated` (for a value of `True`) or `fkData`. This property should be considered read-only.

See also: TField.FieldKind (569)

16.33.37 TField.CanModify

Synopsis: Can the field's contents be modified.

Declaration: `Property CanModify : Boolean`

Visibility: public

Access: Read

Description: `CanModify` is `True` if the field is not read-only and the dataset allows modification.

See also: TField.ReadOnly (573), TDataset.CanModify (519)

16.33.38 TField.CurValue

Synopsis: Current value of the field.

Declaration: `Property CurValue : Variant`

Visibility: public

Access: Read

Description: `CurValue` returns the current value of the field as a variant.

See also: TField.Value (566)

16.33.39 TField.DataSet

Synopsis: Dataset this field belongs to.

Declaration: `Property DataSet : TDataSet`

Visibility: public

Access: Read,Write

Description: `Dataset` contains the dataset this field belongs to. Writing this property will add the field to the list of fields of a dataset, after removing if from the list of fields of the dataset the field was previously assigned to. It should under normal circumstances never be necessary to set this property, the `TDataset` code will take care of this.

See also: TDataset (496), TDataset.Fields (524)

16.33.40 TField.DataSize

Synopsis: Size of the field's data.

Declaration: `Property DataSize : Integer`

Visibility: public

Access: Read

Description: `DataSize` is the memory size needed to store the field's contents. This is different from the Size (565) property which declares a logical size for datatypes that have a variable size (such as string fields). For BLOB fields, use the TBlobField.BlobSize (474) property to get the size of the field's contents for the current record..

See also: TField.Size (565), TBlobField.BlobSize (474)

16.33.41 TField.DataType

Synopsis: The data type of the field.

Declaration: `Property DataType : TFieldType`

Visibility: public

Access: Read

Description: `Datatype` indicates the type of data the field has. This property is initialized when the dataset is opened or when persistent fields are created for the dataset. Instead of checking the class type of the field, it is better to check the `Datatype`, since the actual class of the TField instance may differ depending on the dataset.

See also: TField.FieldKind (569)

16.33.42 TField.DisplayName

Synopsis: User-readable fieldname.

Declaration: `Property DisplayName : string`

Visibility: public

Access: Read

Description: `DisplayName` is the name of the field as it will be displayed to the user e.g. in grid column headers. By default it equals the FieldName (569) property, unless assigned another value.

The use of this property is deprecated. Use DisplayLabel (569) instead.

See also: Tfield.FieldName (569)

16.33.43 TField.DisplayText

Synopsis: Formatted field value.

Declaration: `Property DisplayText : string`

Visibility: public

Access: Read

Description: `DisplayText` returns the field's value as it should be displayed to the user, with all necessary formatting applied. Controls that should display the value of the field should use `DisplayText` instead of the TField.AsString (559) property, which does not take into account any formatting.

See also: TField.AsString (559)

16.33.44 TField.EditMask

Synopsis: Specify an edit mask for an edit control.

Declaration: `Property EditMask : TEditMask`

Visibility: public

Access: Read,Write

Description: `EditMask` can be used to specify an edit mask for controls that allow to edit this field. It has no effect on the field value, and serves only to ensure that the user can enter only correct data for this field.

For more information on valid edit masks, see the documentation of the GUI controls.

See also: TDateTimeField.EditMask (542), TStringField.EditMask (642)

16.33.45 TField.EditMaskPtr

Synopsis: Alias for EditMask.

Declaration: `Property EditMaskPtr : TEditMask`

Visibility: public

Access: Read

Description: `EditMaskPtr` is a read-only alias for the EditMask (563) property. It is not used.

See also: TField.EditMask (563)

16.33.46 TField.FieldNo

Synopsis: Number of the field in the record.

Declaration: `Property FieldNo : LongInt`

Visibility: public

Access: Read

Description: `FieldNo` is the position of the field in the record. It is a 1-based index and is initialized when the dataset is opened or when persistent fields are created for the dataset.

See also: TField.Index (570)

16.33.47 TField.IsIndexField

Synopsis: Is the field an indexed field ?

Declaration: `Property IsIndexField : Boolean`

Visibility: public

Access: Read

Description: `IsIndexField` is true if the field is an indexed field. By default this property is `False`, descendents of TDataset (496) can change this to `True`.

See also: TField.Calculated (561)

16.33.48 TField.IsNull

Synopsis: Is the field empty.

Declaration: `Property IsNull : Boolean`

Visibility: public

Access: Read

Description: `IsNull` is `True` if the field does not have a value. If the underlying data contained a value, or a value is written to it, `IsNull` will return `False`. After TDataset.Insert (512) is called or Clear (553) is called then `IsNull` will return `True`.

See also: TField.Clear (553), TDataset.Insert (512)

16.33.49 TField.NewValue

Synopsis: The new value of the field.

Declaration: `Property NewValue : Variant`

Visibility: public

Access: Read,Write

Description: `NewValue` returns the new value of the field. The FPC implementation of TDataset (496) does not yet support this.

See also: TField.Value (566), TField.CurValue (562)

16.33.50 TField.Offset

Synopsis: Offset of the field's value in the dataset buffer.

Declaration: `Property Offset : Word`

Visibility: public

Access: Read

Description: `Offset` is the location of the field's contents in the dataset memory buffer. It is read-only and initialized by the dataset when it is opened.

See also: TField.FieldNo (564), TField.Index (570), TField.Datasize (562)

16.33.51 TField.Size

Synopsis: Logical size of the field.

Declaration: `Property Size : Integer`

Visibility: public

Access: Read,Write

Description: `Size` is the declared size of the field for datatypes that can have variable size, such as string types, BCD types or array types. To get the size of the storage needed to store the field's content, the DataSize (562) should be used. For blob fields, the current size of the data is not guaranteed to be present.

See also: DataSize (562)

16.33.52 TField.Text

Synopsis: Text representation of the field.

Declaration: `Property Text : string`

Visibility: public

Access: Read,Write

Description: `Text` can be used to retrieve or set the value of the value as a string value for editing purposes. It will trigger the TField.OnGetText (574) event handler if a handler was specified. For display purposes, the TField.DisplayText (563) property should be used. Controls that should display the value in a textual format should use text whenever they must display the text for editing purposes. Inversely, when a control should save the value entered by the user, it should write the contents to the `Text` property, not the AsString (559) property, this will invoke the Tfield.OnSetText (574) event handler, if one is set.

See also: TField.AsString (559), TField.DisplayText (563), TField.Value (566)

16.33.53 TField.ValidChars

Synopsis: Characters that are valid input for the field's content.

Declaration: `Property ValidChars : TFieldChars`

Visibility: public

Access: Read,Write

Description: `ValidChars` is a property that is initialized by descendent classes to contain the set of characters that can be entered in an edit control which is used to edit the field. Numerical fields will set this to a set of numerical characters, string fields will set this to all possible characters. It is possible to restrict the possible input by setting this property to a subset of all possible characters (for example, set it to all uppercase letters to allow the user to enter only uppercase characters. `TField` itself does not enforce the validity of the data when the content of the field is set, an edit control should check the validity of the user input by means of the IsValidChar (554) function.

See also: TField.IsValidChar (554)

16.33.54 TField.Value

Synopsis: Value of the field as a variant value.

Declaration: `Property Value : variant`

Visibility: public

Access: Read,Write

Description: `Value` can be used to read or write the value of the field as a Variant value. When setting the value, the value will be converted to the actual type of the field as defined in the underlying data. Likewise, when reading the value property, the actual field value will be converted to a variant value. If the field does not contain a value (when IsNull (564) returns `True`), then `Value` will contain `Null`.

It is not recommended to use the `Value` property: it should only be used when the type of the field is unknown. If the type of the field is known, it is better to use one of the `AsXXX` properties, which will not only result in faster code, but will also avoid strange type conversions.

See also: TField.IsNull (564), TField.Text (565), TField.DisplayText (563)

16.33.55 TField.OldValue

Synopsis: Old value of the field.

Declaration: `Property OldValue : variant`

Visibility: public

Access: Read

Description: `OldValue` returns the value of the field prior to an edit operation. This feature is currently not supported in FPC.

See also: TField.Value (566), TField.CurValue (562), TField.NewValue (565)

16.33.56 TField.LookupList

Synopsis: List of lookup values.

Declaration: `Property LookupList : TLookupList`

Visibility: public

Access: Read

Description: `LookupList` contains the list of key, value pairs used when caching the possible lookup values for a lookup field. The list is only valid when the LookupCache (571) property is set to `True`. It can be refreshed using the RefreshLookupList (554) method.

See also: TField.RefreshLookupList (554), TField.LookupCache (571)

16.33.57 TField.FieldDef

Synopsis: Fielddef associated with this field.

Declaration: `Property FieldDef : TFieldDef`

Visibility: public

Access: Read

Description: `FieldDef` references the `TFieldDef` instance to which this field instance is bound. When a dataset is opened, the TDataset.FieldDefs (521) property is filled with field definitions as returned from the server. After this fields are created, or if they already exist, are bound to these fielddefs.

See also: TDataset.FieldDefs (521)

16.33.58 TField.Alignment

Synopsis: Alignment for this field.

Declaration: `Property Alignment : TAlignment`

Visibility: published

Access: Read,Write

Description: `Alignment` contains the alignment that UI controls should observe when displaying the contents of the field. Setting the property at the field level will make sure that all DB-Aware controls will display the contents of the field with the same alignment.

See also: TField.DisplayText (563)

16.33.59 TField.CustomConstraint

Synopsis: Custom constraint for the field's value.

Declaration: `Property CustomConstraint : string`

Visibility: published

Access: Read,Write

Description: `CustomConstraint` may contain a constraint that will be enforced when the dataset posts it's data. It should be a SQL-like expression that results in a `True` or `False` value. Examples of valid constraints are:

```
Salary < 10000
YearsEducation < Age
```

If the constraint is not satisfied when the record is posted, then an exception will be raised with the value of ConstraintErrorMessage (568) as a message.

This feature is not yet implemented in FPC.

See also: TField.ConstraintErrorMessage (568), TField.ImportedConstraint (570)

16.33.60 TField.ConstraintErrorMessage

Synopsis: Message to display if the `CustomConstraint` constraint is violated.

Declaration: `Property ConstraintErrorMessage : string`

Visibility: published

Access: Read,Write

Description: `ConstraintErrorMessage` is the message that should be displayed when the dataset checks the constraints and the constraint in TField.CustomConstraint (568) is violated.

This feature is not yet implemented in FPC.

See also: TField.CustomConstraint (568)

16.33.61 TField.DefaultExpression

Synopsis: Default value for the field.

Declaration: `Property DefaultExpression : string`

Visibility: published

Access: Read,Write

Description: `DefaultValue` can be set to a value that should be entered in the field whenever the TDataset.Append (503) or TDataset.Insert (512) methods are executed. It should contain a valid SQL expression that results in the correct type for the field.

This feature is not yet implemented in FPC.

See also: TDataset.Insert (512), TDataset.Append (503), TDataset.CustomConstraint (496)

16.33.62 TField.DisplayLabel

Synopsis: Name of the field for display purposes.

Declaration: `Property DisplayLabel : string`

Visibility: published

Access: Read,Write

Description: `DisplayLabel` is the name of the field as it will be displayed to the user e.g. in grid column headers. By default it equals the FieldName (569) property, unless assigned another value.

See also: TField.FieldName (569)

16.33.63 TField.DisplayWidth

Synopsis: Width of the field in characters.

Declaration: `Property DisplayWidth : LongInt`

Visibility: published

Access: Read,Write

Description: `DisplayWidth` is the width (in characters) that should be used by controls that display the contents of the field (such as in grids or lookup lists). It is initialized to a default value for most fields (e.g. it equals Size (565) for string fields) but can be modified to obtain a more appropriate value for the field's expected content.

See also: TField.Alignment (567), TField.DisplayText (563)

16.33.64 TField.FieldKind

Synopsis: The kind of field.

Declaration: `Property FieldKind : TFieldKind`

Visibility: published

Access: Read,Write

Description: `FieldKind` indicates the type of the `TField` instance. Besides `TField` instances that represent fields present in the underlying data records, there can also be calculated or lookup fields. This property determines what kind of field the TField instance is.

16.33.65 TField.FieldName

Synopsis: Name of the field.

Declaration: `Property FieldName : string`

Visibility: published

Access: Read,Write

Description: `FieldName` is the name of the field as it is defined in the underlying data structures (for instance the name of the field in a SQL table, DBAse file, or the alias of the field if it was aliased in a SQL SELECT statement. It does not always equal the `Name` property, which is the name of the `TField` component instance. The `Name` property will generally equal the name of the dataset appended with the value of the `FieldName` property.

See also: TFieldDef.Name (575), TField.Size (565), TField.DataType (562)

16.33.66 TField.HasConstraints

Synopsis: Does the field have any constraints defined.

Declaration: `Property HasConstraints : Boolean`

Visibility: published

Access: Read

Description: `HasConstraints` will contain `True` if one of the CustomConstraint (568) or ImportedConstraint (570) properties is set to a non-empty value.

See also: CustomConstraint (568), ImportedConstraint (570)

16.33.67 TField.Index

Synopsis: Index of the field in the list of fields.

Declaration: `Property Index : LongInt`

Visibility: published

Access: Read,Write

Description: `Index` is the name of the field in the list of fields of a dataset. It is, in general, the (0-based) position of the field in the underlying data structures, but this need not always be so. The TField.FieldNo (564) property should be used for that.

See also: TField.FieldNo (564)

16.33.68 TField.ImportedConstraint

Synopsis: Constraint for the field value on the level of the underlying database.

Declaration: `Property ImportedConstraint : string`

Visibility: published

Access: Read,Write

Description: `ImportedConstraint` contains any constraints that the underlying data engine imposes on the values of a field (usually in an SQL CONSTRAINT) clause. Whether this field is filled with appropriate data depends on the implementation of the TDataset (496) descendent.

See also: TField.CustomConstraint (568), TDataset (496), TField.ConstraintErrorMessage (568)

16.33.69 TField.KeyFields

Synopsis: Key fields to use when looking up a field value.

Declaration: `Property KeyFields : string`

Visibility: published

Access: Read,Write

Description: `KeyFields` should contain a semi-colon separated list of field names from the lookupfield's dataset which will be matched to the fields enumerated in LookupKeyFields (571) in the dataset pointed to by the LookupDataset (571) property.

See also: LookupKeyFields (571), LookupDataset (571)

16.33.70 TField.LookupCache

Synopsis: Should lookup values be cached.

Declaration: `Property LookupCache : Boolean`

Visibility: published

Access: Read,Write

Description: `LookupCache` is by default `False`. If it is set to `True` then a list of key, value pairs will be created from the LookupKeyFields (571) in the dataset pointed to by the LookupDataset (571) property. The list of key, value pairs is available through the TField.LookupList (567) property.

See also: LookupKeyFields (571), LookupDataset (571), TField.LookupList (567)

16.33.71 TField.LookupDataSet

Synopsis: Dataset with lookup values.

Declaration: `Property LookupDataSet : TDataSet`

Visibility: published

Access: Read,Write

Description: `LookupDataset` is used by lookup fields to fetch the field's value. The LookupKeyFields (571) property is used as a list of fields to locate a record in this dataset, and the value of the LookupResultField (572) field is then used as the value of the lookup field.

See also: KeyFields (571), LookupKeyFields (571), LookupResultField (572), LookupCache (571)

16.33.72 TField.LookupKeyFields

Synopsis: Names of fields on which to perform a locate.

Declaration: `Property LookupKeyFields : string`

Visibility: published

Access: Read,Write

Description: `LookupKeyFields` should contain a semi-colon separated list of field names from the dataset pointed to by the LookupDataset (571) property. These fields will be used when locating a record corresponding to the values in the TField.KeyFields (571) property.

See also: KeyFields (571), LookupDataset (571), LookupResultField (572), LookupCache (571)

16.33.73 TField.LookupResultField

Synopsis: Name of field to use as lookup value.

Declaration: `Property LookupResultField : string`

Visibility: published

Access: Read,Write

Description: `LookupResultField` contains the field name from a field in the dataset pointed to by the LookupDataset (571) property. The value of this field will be used as the lookup's field value when a record is found in the lookup dataset as result for the lookup field value.

See also: KeyFields (571), LookupDataset (571), LookupKeyFields (571), LookupCache (571)

16.33.74 TField.Lookup

Synopsis: Is the field a lookup field.

Declaration: `Property Lookup : Boolean; deprecated;`

Visibility: published

Access: Read,Write

Description: `Lookup` is `True` if the FieldKind (569) equals `fkLookup`, `False` otherwise. Setting the `Lookup` property will switch the `FieldKind` between the `fkLookup` and `fkData`.

See also: TField.FieldKind (569)

16.33.75 TField.Origin

Synopsis: Original fieldname of the field.

Declaration: `Property Origin : string`

Visibility: published

Access: Read,Write

Description: `Origin` contains the origin of the field in the form `TableName.fieldName`. This property is filled only if the TDataset (496) descendent or the database engine support retrieval of this property. It can be used to automatically create update statements, together with the TField.ProviderFlags (573) property.

See also: TDataset (496), TField.ProviderFlags (573)

16.33.76 TField.ParentField

Declaration: `Property ParentField : TObjectField`

Visibility: published

Access: Read,Write

16.33.77 TField.ProviderFlags

Synopsis: Flags for provider or update support.

Declaration: `Property ProviderFlags : TProviderFlags`

Visibility: published

Access: Read,Write

Description: `ProviderFlags` contains a set of flags that can be used by engines that automatically generate update SQL statements or update data packets. The various items in the set tell the engine whether the key is a key field, should be used in the where clause of an update statement or whether - in fact - it should be updated at all.

These properties should be set by the programmer so engines such as SQLDB can create correct update SQL statements whenever they need to post changes to the database. Note that to be able to set these properties in a designer, persistent fields must be created.

See also: TField.Origin (572)

16.33.78 TField.ReadOnly

Synopsis: Is the field read-only.

Declaration: `Property ReadOnly : Boolean`

Visibility: published

Access: Read,Write

Description: `ReadOnly` can be set to `True` to prevent controls of writing data to the field, effectively making it a read-only field. Setting this property to `True` does not prevent the field from getting a value through code: it is just an indication for GUI controls that the field's value is considered read-only.

See also: TFieldDef.Attributes (579)

16.33.79 TField.Required

Synopsis: Does the field require a value.

Declaration: `Property Required : Boolean`

Visibility: published

Access: Read,Write

Description: `Required` determines whether the field needs a value when posting the data: when a dataset posts the changed made to a record (new or existing), it will check whether all fields with the `Required` property have a value assigned to them. If not, an exception will be raised. Descendents of TDataset (496) will set the property to `True` when opening the dataset, depending on whether the field is required in the underlying data engine. For fields that are not required by the database engine, the programmer can still set the property to `True` if the business logic requires a field.

See also: TDataset.Open (515), ReadOnly (573), Visible (574)

16.33.80 TField.Visible

Synopsis: Should the field be shown in grids.

Declaration: `Property Visible : Boolean`

Visibility: published

Access: Read,Write

Description: `Visible` can be used to hide fields from a grid when displaying data to the user. Invisible fields will by default not be shown in the grid.

See also: TField.ReadOnly (573), TField.Required (573)

16.33.81 TField.OnChange

Synopsis: Event triggered when the field's value has changed.

Declaration: `Property OnChange : TFieldNotifyEvent`

Visibility: published

Access: Read,Write

Description: `OnChange` is triggered whenever the field's value has been changed. It is triggered only after the new contents have been written to the dataset buffer, so it can be used to react to changes in the field's content. To prevent the writing of changes to the buffer, use the TField.OnValidate (575) event. It is not allowed to change the state of the dataset or the contents of the field during the execution of this event handler: doing so may lead to infinite loops and other unexpected results.

See also: TField.OnChange (574)

16.33.82 TField.OnGetText

Synopsis: Event to format the field's content.

Declaration: `Property OnGetText : TFieldGetTextEvent`

Visibility: published

Access: Read,Write

Description: `OnGetText` is triggered whenever the TField.Text (565) or TField.DisplayText (563) properties are read. It can be used to return a custom formatted string in the `AText` parameter which will then typically be used by a control to display the field's contents to the user. It is not allowed to change the state of the dataset or the contents of the field during the execution of this event handler.

See also: TField.Text (565), TField.DisplayText (563), TField.OnSetText (574), TFieldGetTextEvent (445)

16.33.83 TField.OnSetText

Synopsis: Event to set the field's content based on a user-formatted string.

Declaration: `Property OnSetText : TFieldSetTextEvent`

Visibility: published

Access: Read,Write

Description: `OnSetText` is called whenever the TField.Text (565) property is written. It can be used to set the actual value of the field based on the passed `AText` parameter. Typically, this event handler will perform the inverse operation of the TField.OnGetText (574) handler, if it exists.

See also: TField.Text (565), TField.OnGetText (574), TFieldGetTextEvent (445)

16.33.84 TField.OnValidate

Synopsis: Event to validate the value of a field before it is written to the data buffer.

Declaration: `Property OnValidate : TFieldNotifyEvent`

Visibility: published

Access: Read,Write

Description: `OnValidate` is called prior to writing a new field value to the dataset's data buffer. It can be used to prevent writing the new value to the buffer by raising an exception in the event handler. Note that this event handler is always called, irrespective of the way the value of the field is set.

See also: TField.Text (565), TField.OnGetText (574), TField.OnSetText (574), TField.OnChange (574)

16.34 TFieldDef

16.34.1 Description

`TFieldDef` is used to describe the fields that are present in the data underlying the dataset. For each field in the underlying field, an `TFieldDef` instance is created when the dataset is opened. This class offers almost no methods, it is mainly a storage class, to store all relevant properties of fields in a record (name, data type, size, required or not, etc.)

See also: TDataset.FieldDefs (521), TFieldDefs (581)

16.34.2 Method overview

Page	Method	Description
577	AddChild	
577	Assign	Assign the contents of one TFieldDef instance to another.
576	Create	Constructor for `TFieldDef`.
577	CreateField	Create `TField` instance based on definitions in current `TFieldDef` instance.
576	Destroy	Free the TFieldDef instance.
577	HasChildDefs	

16.34.3 Property overview

Page	Properties	Access	Description
579	Attributes	rw	Additional attributes of the field.
578	CharSize	r	Character size.
580	ChildDefs	rws	
579	Codepage	r	System code page for the values in string and wide string field types.
580	DataType	rw	Data type for the field.
577	FieldClass	r	TField class used for this fielddef.
578	FieldNo	r	Field number.
578	InternalCalcField	rw	Is this a definition of an internally calculated field ?
579	ParentDef	r	
580	Precision	rw	Precision used in BCD (Binary Coded Decimal) fields.
579	Required	rw	Is the field required ?
580	Size	rw	Size of the buffer needed to store the data of the field.

16.34.4 TFieldDef.Create

Synopsis: Constructor for `TFieldDef`.

Declaration:
```
constructor Create(ACollection: TCollection);  Override
constructor Create(AOwner: TFieldDefs; const AName: string;
                   ADataType: TFieldType; ASize: Integer;
                   ARequired: Boolean; AFieldNo: LongInt;
                   ACodePage: TSystemCodePage);  Overload
```

Visibility: public

Description: `Create` is the constructor for the `TFieldDef` class.

If a simple call is used, with a single argument `ACollection`, the inherited `Create` is called and the Field number is set to the incremented current index.

If the more complicated call is used, with multiple arguments, then after the inherited `Create` call, the Name (575), datatype (580), size (580), precision (580), FieldNo (578), Required (579) and CodePage (579) property are all set according to the passed arguments.

Errors: If a duplicate name is passed, then an exception will occur.

See also: Name (575), datatype (580), size (580), precision (580), FieldNo (578), Required (579), CodePage (579)

16.34.5 TFieldDef.Destroy

Synopsis: Free the TFieldDef instance.

Declaration:
```
destructor Destroy;  Override
```

Visibility: public

Description: `Destroy` destroys the TFieldDef instance. It simply calls the inherited destructor.

See also: TFieldDef.Create (576)

16.34.6 TFieldDef.AddChild

Declaration: `function AddChild : TFieldDef`

Visibility: public

16.34.7 TFieldDef.Assign

Synopsis: Assign the contents of one TFieldDef instance to another.

Declaration: `procedure Assign(APersistent: TPersistent);  Override`

Visibility: public

Description: `Assign` assigns all published properties of `APersistent` to the current instance, if `APersistent` is an instance of class `TFieldDef`.

Errors: If `APersistent` is not of class TFieldDef (575), then an exception will be raised.

16.34.8 TFieldDef.CreateField

Synopsis: Create `TField` instance based on definitions in current `TFieldDef` instance.

Declaration:
```
function CreateField(AOwner: TComponent; ParentField: TObjectField;
                     const FieldName: string; CreateChildren: Boolean)
                      : TField
```

Visibility: public

Description: `CreateField` determines, based on the DataType (580) what TField (549) descendent it should create, and then returns a newly created instance of this class. It sets the appropriate defaults for the Size (565), FieldName (569), FieldNo (564), Precision (549), ReadOnly (573) and Required (573) properties of the newly created instance. It should never be necessary to use this call in an end-user program, only `TDataset` descendent classes should use this call.

The newly created field is owned by the component instance passed in the `AOwner` parameter.

The DefaultFieldClasses (439) array is used to determine which TField Descendent class should be used when creating the TField instance, but descendents of TDataset may override the values in that array.

See also: DefaultFieldClasses (439), TField (549)

16.34.9 TFieldDef.HasChildDefs

Declaration: `function HasChildDefs : Boolean`

Visibility: public

16.34.10 TFieldDef.FieldClass

Synopsis: TField class used for this fielddef.

Declaration: `Property FieldClass : TFieldClass`

Visibility: public

Access: Read

Description: `FieldClass` is the class of the `TField` instance that is created by the CreateField (577) class. The return value is retrieved from the `TDataset` instance the `TFieldDef` instance is associated with. If there is no TDataset instance available, the return value is `Nil`

See also: TDataset (496), CreateField (577), TField (549)

16.34.11 TFieldDef.FieldNo

Synopsis: Field number.

Declaration: `Property FieldNo : LongInt`

Visibility: public

Access: Read

Description: `FieldNo` is the number of the field in the data structure where the dataset contents comes from, for instance in a DBase file. If the underlying data layer does not support the concept of field number, a sequential number is assigned.

16.34.12 TFieldDef.CharSize

Synopsis: Character size.

Declaration: `Property CharSize : Word`

Visibility: public

Access: Read

Description: `CharSize` is only relevant for for string fields: it indicates the number of bytes used to represent a single character. It is calculated from the TFieldDef.CodePage (579) property and can have the following values:

- 1 for single-byte string fields
- 2 for UnicodeString fields
- 4 for UTF8 strings

See also: TFieldDef.CodePage (579), TFieldDef.Size (580)

16.34.13 TFieldDef.InternalCalcField

Synopsis: Is this a definition of an internally calculated field ?

Declaration: `Property InternalCalcField : Boolean`

Visibility: public

Access: Read,Write

Description: `Internalcalc` is `True` if the fielddef instance represents an internally calculated field: for internally calculated fields, storage must be provided by the underlying data mechanism.

16.34.14 TFieldDef.ParentDef

Declaration: `Property ParentDef : TFieldDef`

Visibility: public

Access: Read

16.34.15 TFieldDef.Required

Synopsis: Is the field required ?

Declaration: `Property Required : Boolean`

Visibility: public

Access: Read,Write

Description: `Required` is set to `True` if the field requires a value when posting data to the dataset. If no value was entered, the dataset will raise an exception when the record is posted. The `Required` property is usually initialized based on the definition of the field in the underlying database. For SQL-based databases, a field declared as `NOT NULL` will result in a `Required` property of `True`.

16.34.16 TFieldDef.Codepage

Synopsis: System code page for the values in string and wide string field types.

Declaration: `Property Codepage : TSystemCodePage`

Visibility: public

Access: Read

Description: Codepage is a read-only TSystemCodePage property with the system code page used for values in the field. The value in CodePage is assigned in the overloaded constructor which includes a TSystemCodePage argument. CodePage is relevant for string or wide string field types, and uses the following values:

ftString, ftFixedChar, ftMemo Uses the value passed in the argument. The default value is 0.

ftWideString, ftFixedWideChar, ftWideMemo Use the value in the CP_UTF16 constant.

Other non-string field types Uses the value 0 in Codepage.

See also: TFieldDef.Create (576), TFieldDef.DataType (580)

16.34.17 TFieldDef.Attributes

Synopsis: Additional attributes of the field.

Declaration: `Property Attributes : TFieldAttributes`

Visibility: published

Access: Read,Write

Description: `Attributes` contain additional attributes of the field. It shares the `faRequired` attribute with the `Required` property.

See also: TFieldDef.Required (579)

16.34.18 TFieldDef.DataType

Synopsis: Data type for the field.

Declaration: `Property DataType : TFieldType`

Visibility: published

Access: Read,Write

Description: `DataType` contains the data type of the field's contents. Based on this property, the `FieldClass` property determines what kind of field class must be used to represent this field.

See also: TFieldDef.FieldClass (577), TFieldDef.CreateField (577)

16.34.19 TFieldDef.ChildDefs

Declaration: `Property ChildDefs : TFieldDefs`

Visibility: published

Access: Read,Write

16.34.20 TFieldDef.Precision

Synopsis: Precision used in BCD (Binary Coded Decimal) fields.

Declaration: `Property Precision : LongInt`

Visibility: published

Access: Read,Write

Description: `Precision` is the number of digits used in a BCD (Binary Coded Decimal) field. It is not the number of digits after the decimal separator, but the total number of digits.

See also: TFieldDef.Size (580)

16.34.21 TFieldDef.Size

Synopsis: Size of the buffer needed to store the data of the field.

Declaration: `Property Size : Integer`

Visibility: published

Access: Read,Write

Description: `Size` indicates the size of the buffer needed to hold data for the field. For types with a fixed size (such as integer, word or data/time) the size can be zero: the buffer mechanism reserves automatically enough heap memory. For types which can have various sizes (blobs, string types), the Size property tells the buffer mechanism how many bytes are needed to hold the data for the field. For BCD fields, the size property indicates the number of decimals after the decimal separator.

See also: TFieldDef.Precision (580), TFieldDef.DataType (580)

16.35 TFieldDefs

16.35.1 Description

`TFieldDefs` is used by each `TDataset` instance to keep a description of the data that it manages; for each field in a record that makes up the underlying data, the `TFieldDefs` instance keeps an instance of `TFieldDef` that describes the field's contents. For any internally calculated fields of the dataset, a `TFieldDef` instance is kept as well. This collection is filled by descendent classes of `TDataset` as soon as the dataset is opened; it is cleared when the dataset closes. After the collection was populated, the dataset creates `TField` instances based on all the definitions in the collections. If persistent fields were used, the contents of the fielddefs collection is compared to the field components that are present in the dataset. If the collection contains more field definitions than Field components, these extra fields will not be available in the dataset.

See also: TFieldDef (575), TDataset (496)

16.35.2 Method overview

Page	Method	Description
581	Add	Add a new field definition to the collection.
582	AddFieldDef	Add new `TFieldDef`.
582	Assign	Copy all items from one dataset to another.
581	Create	Create a new instance of `TFieldDefs`.
582	Find	Find item by name.
583	MakeNameUnique	Create a unique field name starting from a base name.
583	Update	Force update of definitions.

16.35.3 Property overview

Page	Properties	Access	Description
583	HiddenFields	rw	Should field instances be created for hidden fields.
583	Items	rw	Indexed access to the fielddef instances.
584	ParentDef	r	

16.35.4 TFieldDefs.Create

Synopsis: Create a new instance of `TFieldDefs`.

Declaration:
```
constructor Create(AOwner: TPersistent)
```

Visibility: public

Description: `Create` is used to create a new instance of `TFieldDefs`. The `ADataset` argument contains the dataset instance for which the collection contains the field definitions.

See also: TFieldDef (575), TDataset (496)

16.35.5 TFieldDefs.Add

Synopsis: Add a new field definition to the collection.

Declaration:
```
function Add(const AName: string; ADataType: TFieldType;
                  ASize: Integer; APrecision: Integer; ARequired: Boolean;
                  AReadOnly: Boolean; AFieldNo: Integer;
```

```
            ACodePage: TSystemCodePage) : TFieldDef;  Overload
function Add(const AName: string; ADataType: TFieldType; ASize: Word;
            ARequired: Boolean; AFieldNo: Integer) : TFieldDef
            ;  Overload
procedure Add(const AName: string; ADataType: TFieldType; ASize: Word;
             ARequired: Boolean);  Overload
procedure Add(const AName: string; ADataType: TFieldType; ASize: Word)
             ;  Overload
procedure Add(const AName: string; ADataType: TFieldType);  Overload
```

Visibility: public

Description: `Add` adds a new item to the collection and fills in the `Name`, `DataType`, `Size` and `Required` properties of the newly added item with the provided parameters.

Errors: If an item with name `AName` already exists in the collection, then an exception will be raised.

See also: TFieldDefs.AddFieldDef (582)

16.35.6 TFieldDefs.AddFieldDef

Synopsis: Add new `TFieldDef`.

Declaration: `function AddFieldDef : TFieldDef`

Visibility: public

Description: `AddFieldDef` creates a new `TFieldDef` item and returns the instance.

See also: TFieldDefs.Add (581)

16.35.7 TFieldDefs.Assign

Synopsis: Copy all items from one dataset to another.

Declaration: `procedure Assign(FieldDefs: TFieldDefs);  Overload`

Visibility: public

Description: `Assign` simply calls inherited `Assign` with the `FieldDefs` argument.

See also: TFieldDef.Assign (577)

16.35.8 TFieldDefs.Find

Synopsis: Find item by name.

Declaration: `function Find(const AName: string) : TFieldDef`

Visibility: public

Description: `Find` simply calls the inherited TDefCollection.Find (546) to find an item with name `AName` and typecasts the result to `TFieldDef`.

See also: TDefCollection.Find (546), TNamedItem.Name (614)

16.35.9 TFieldDefs.Update

Synopsis: Force update of definitions.

Declaration: `procedure Update;  Overload`

Visibility: public

Description: `Update` notifies the dataset that the field definitions are updated, if it was not yet notified.

See also: TDefCollection.Updated (547)

16.35.10 TFieldDefs.MakeNameUnique

Synopsis: Create a unique field name starting from a base name.

Declaration: `function MakeNameUnique(const AName: string) : string;  Virtual`

Visibility: public

Description: `MakeNameUnique` uses `AName` to construct a name of a field that is not yet in the collection. If `AName` is not yet in the collection, then `AName` is returned. if a field definition with field name equal to `AName` already exists, then a new name is constructed by appending a sequence number to `AName` till the resulting name does not appear in the list of field definitions.

See also: TFieldDefs.Find (582), TFieldDef.Name (575)

16.35.11 TFieldDefs.HiddenFields

Synopsis: Should field instances be created for hidden fields.

Declaration: `Property HiddenFields : Boolean`

Visibility: public

Access: Read,Write

Description: `HiddenFields` determines whether a field is created for fielddefs that have the `faHiddenCol` attribute set. If set to `False` (the default) then no `TField` instances will be created for hidden fields. If it is set to `True`, then a TField instance will be created for hidden fields.

See also: TFieldDef.Attributes (579)

16.35.12 TFieldDefs.Items

Synopsis: Indexed access to the fielddef instances.

Declaration: `Property Items[Index: LongInt]: TFieldDef; default`

Visibility: public

Access: Read,Write

Description: `Items` provides zero-based indexed access to all `TFieldDef` instances in the collection. The index must vary between 0 and `Count-1`, or an exception will be raised.

See also: TFieldDef (575)

16.35.13 TFieldDefs.ParentDef

Declaration: `Property ParentDef : TFieldDef`

Visibility: public

Access: Read

16.36 TFields

16.36.1 Description

`TFields` mimics a `TCollection` class for the Fields (524) property of TDataset (496) instance. Since TField (549) is a descendent of `TComponent`, it cannot be an item of a collection, and must be managed by another class.

See also: TField (549), TDataset (496), TDataset.Fields (524)

16.36.2 Method overview

Page	Method	Description
585	Add	Add a new field to the list.
585	CheckFieldName	Check field name for duplicate entries.
585	CheckFieldNames	Check a list of field names for duplicate entries.
586	Clear	Clear the list of fields.
584	Create	Create a new instance of `TFields`.
585	Destroy	Free the `TFields` instance.
586	FieldByName	Find a field based on its name.
586	FieldByNumber	Search field based on its fieldnumber.
586	FindField	Find a field based on its name.
587	GetEnumerator	Return an enumerator for the `for..in` construct.
587	GetFieldNames	Get the list of fieldnames.
587	IndexOf	Return the index of a field instance.
587	Remove	Remove an instance from the list.

16.36.3 Property overview

Page	Properties	Access	Description
588	Count	r	Number of fields in the list.
588	Dataset	r	Dataset the fields belong to.
588	Fields	rw	Indexed access to the fields in the list.

16.36.4 TFields.Create

Synopsis: Create a new instance of `TFields`.

Declaration: `constructor Create(ADataset: TDataSet)`

Visibility: public

Description: `Create` initializes a new instance of `TFields`. It stores the `ADataset` parameter, so it can be retrieved at any time in the TFields.Dataset (588) property, and initializes an internal list object to store the list of fields.

See also: TDataset (496), TFields.Dataset (588), TField (549)

16.36.5 TFields.Destroy

Synopsis: Free the `TFields` instance.

Declaration: `destructor Destroy;  Override`

Visibility: public

Description: `Destroy` frees the field instances that it manages on behalf of the Dataset (588). After that it cleans up the internal structures and then calls the inherited destructor.

See also: TDataset (496), TField (549), TFields.Clear (586)

16.36.6 TFields.Add

Synopsis: Add a new field to the list.

Declaration: `procedure Add(Field: TField)`

Visibility: public

Description: `Add` must be used to add a new TField (549) instance to the list of fields. After a `TField` instance is added to the list, the `TFields` instance will free the field instance if it is cleared.

See also: TField (549), TFields.Clear (586)

16.36.7 TFields.CheckFieldName

Synopsis: Check field name for duplicate entries.

Declaration: `procedure CheckFieldName(const Value: string)`

Visibility: public

Description: `CheckFieldName` checks whether a field with name equal to `Value` (case insensitive) already appears in the list of fields (using TFields.Find (584)). If it does, then an EDatabaseError (458) exception is raised.

See also: TField.FieldName (569), TFields.Find (584)

16.36.8 TFields.CheckFieldNames

Synopsis: Check a list of field names for duplicate entries.

Declaration: `procedure CheckFieldNames(const Value: string)`

Visibility: public

Description: `CheckFieldNames` splits `Value` in a list of fieldnames, using semicolon as a separator. For each of the fieldnames obtained in this way, it calls CheckFieldName (585).

Errors: Spaces are not discarded, so leaving a space after of before a fieldname will not find the fieldname, and will yield a false negative result.

See also: TField.FieldName (569), TFields.CheckFieldName (585), TFields.Find (584)

16.36.9 TFields.Clear

Synopsis: Clear the list of fields.

Declaration: `procedure Clear`

Visibility: public

Description: `Clear` removes all TField (549) var instances from the list. All field instances are freed after they have been removed from the list.

See also: TField (549)

16.36.10 TFields.FindField

Synopsis: Find a field based on its name.

Declaration: `function FindField(const Value: string) : TField`

Visibility: public

Description: `FindField` searches the list of fields and returns the field instance whose FieldName (569) property matches `Value`. The search is performed case-insensitively. If no field instance is found, then `Nil` is returned.

See also: TFields.FieldByName (586)

16.36.11 TFields.FieldByName

Synopsis: Find a field based on its name.

Declaration: `function FieldByName(const Value: string) : TField`

Visibility: public

Description: `Fieldbyname` searches the list of fields and returns the field instance whose FieldName (569) property matches `Value`. The search is performed case-insensitively.

Errors: If no field instance is found, then an exception is raised. If this behaviour is undesired, use TField.FindField (549), where `Nil` is returned if no match is found.

See also: TFields.FindField (586), TFields.FieldName (584), Tfields.FieldByNumber (586), TFields.IndexOf (587)

16.36.12 TFields.FieldByNumber

Synopsis: Search field based on its fieldnumber.

Declaration: `function FieldByNumber(FieldNo: Integer) : TField`

Visibility: public

Description: `FieldByNumber` searches for the field whose TField.FieldNo (564) property matches the `FieldNo` parameter. If no such field is found, `Nil` is returned.

See also: TFields.FieldByName (586), TFields.FindField (586), TFields.IndexOf (587)

16.36.13 TFields.GetEnumerator

Synopsis: Return an enumerator for the `for..in` construct.

Declaration: `function GetEnumerator : TFieldsEnumerator`

Visibility: public

Description: `GetEnumerator` is the implementation of `IEnumerable` and returns an instance of TFieldsEnumerator (588)

See also: TFieldsEnumerator (588), #rtl.system.IEnumerable (**??**)

16.36.14 TFields.GetFieldNames

Synopsis: Get the list of fieldnames.

Declaration: `procedure GetFieldNames(Values: TStrings)`

Visibility: public

Description: `GetFieldNames` fills `Values` with the fieldnames of all the fields in the list, each item in the list contains 1 fieldname. The list is cleared prior to filling it.

See also: TField.FieldName (569)

16.36.15 TFields.IndexOf

Synopsis: Return the index of a field instance.

Declaration: `function IndexOf(Field: TField) : LongInt`

Visibility: public

Description: `IndexOf` scans the list of fields and returns the index of the field instance in the list (it compares actual field instances, not field names). If the field does not appear in the list, -1 is returned.

See also: TFields.FieldByName (586), TFields.FieldByNumber (586), TFields.FindField (586)

16.36.16 TFields.Remove

Synopsis: Remove an instance from the list.

Declaration: `procedure Remove(Value: TField)`

Visibility: public

Description: `Remove` removes the field `Value` from the list. It does not free the field after it was removed. If the field is not in the list, then nothing happens.

See also: TFields.Clear (586)

16.36.17 TFields.Count

Synopsis: Number of fields in the list.

Declaration: `Property Count : Integer`

Visibility: public

Access: Read

Description: `Count` is the number of fields in the fieldlist. The items in the Fields (588) property are numbered from 0 to `Count-1`.

See also: TFields.fields (588)

16.36.18 TFields.Dataset

Synopsis: Dataset the fields belong to.

Declaration: `Property Dataset : TDataSet`

Visibility: public

Access: Read

Description: `Dataset` is the dataset instance that owns the fieldlist. It is set when the TFields (584) instance is created. This property is purely for informational purposes. When adding fields to the list, no check is performed whether the field's `Dataset` property matches this dataset.

See also: TFields.Create (584), TField.Dataset (562), TDataset (496)

16.36.19 TFields.Fields

Synopsis: Indexed access to the fields in the list.

Declaration: `Property Fields[Index: Integer]: TField; default`

Visibility: public

Access: Read,Write

Description: `Fields` is the default property of the `TFields` class. It provides indexed access to the fields in the list: the index runs from 0 to `Count-1`.

Errors: Providing an index outside the allowed range will result in an `EListError` exception.

See also: TFields.FieldByName (586)

16.37 TFieldsEnumerator

16.37.1 Description

`TFieldsEnumerator` implements all the methods of `IEnumerator` so a TFields (584) instance can be used in a for..in construct. `TFieldsEnumerator` returns all the fields in the `TFields` collection. Therefor the following construct is possible:

```
Var
  F : TField;

begin
   // ...
   For F in MyDataset.Fields do
     begin
     // F is of type TField.
     end;
   // ...
```

Do not create an instance of `TFieldsEnumerator` manually. The compiler will do all that is needed when it encounters the for..in construct.

See also: TField (549), TFields (584), #rtl.system.IEnumerator (**??**)

16.37.2 Method overview

Page	Method	Description
589	Create	Create a new instance of `TFieldsEnumerator`.
589	MoveNext	Move the current field to the next field in the collection.

16.37.3 Property overview

Page	Properties	Access	Description
590	Current	r	Return the current field.

16.37.4 TFieldsEnumerator.Create

Synopsis: Create a new instance of `TFieldsEnumerator`.

Declaration: `constructor Create(AFields: TFields)`

Visibility: public

Description: `Create` instantiates a new instance of `TFieldsEnumerator`. It stores the `AFields` reference, pointing to the TFields (584) instance that created the enumerator. It initializes the enumerator position.

16.37.5 TFieldsEnumerator.MoveNext

Synopsis: Move the current field to the next field in the collection.

Declaration: `function MoveNext : Boolean`

Visibility: public

Description: `MoveNext` moves the internal pointer to the next field in the fields collection, and returns `True` if the operation was a success. If no more fields are available, then `False` is returned.

See also: TFieldsEnumerator.Current (590)

16.37.6 TFieldsEnumerator.Current

Synopsis: Return the current field.

Declaration: `Property Current : TField`

Visibility: public

Access: Read

Description: `Current` returns the current field. It will return a non-nil value only after `MoveNext` returned `True`.

See also: TFieldsEnumerator.MoveNext (589)

16.38 TFloatField

16.38.1 Description

`TFloatField` is the class created when a dataset must manage floating point values of double precision. It exposes a few new properties such as Currency (591), MaxValue (591), MinValue (592) and overrides some TField (549) methods to work with floating point data.

It should never be necessary to create an instance of `TFloatField` manually, a field of this class will be instantiated automatically for each floating-point field when a dataset is opened.

See also: Currency (591), MaxValue (591), MinValue (592)

16.38.2 Method overview

Page	Method	Description
591	CheckRange	Check whether a value is in the allowed range of values for the field.
590	Create	Create a new instance of the `TFloatField`.

16.38.3 Property overview

Page	Properties	Access	Description
591	Currency	rw	Is the field a currency field.
591	MaxValue	rw	Maximum value for the field.
592	MinValue	rw	Minimum value for the field.
592	Precision	rw	Precision (number of digits) of the field in text representations.
591	Value	rw	Value of the field as a double type.

16.38.4 TFloatField.Create

Synopsis: Create a new instance of the `TFloatField`.

Declaration: `constructor Create(AOwner: TComponent);  Override`

Visibility: public

Description: `Create` initializes a new instance of `TFloatField`. It calls the inherited constructor and then initializes some properties.

16.38.5 TFloatField.CheckRange

Synopsis: Check whether a value is in the allowed range of values for the field.

Declaration: `function CheckRange(AValue: Double) : Boolean`

Visibility: public

Description: `CheckRange` returns `True` if `AValue` lies within the range defined by the MinValue (592) and MaxValue (591) properties. If the value lies outside of the allowed range, then `False` is returned.

See also: MaxValue (591), MinValue (592)

16.38.6 TFloatField.Value

Synopsis: Value of the field as a double type.

Declaration: `Property Value : Double`

Visibility: public

Access: Read,Write

Description: `Value` is redefined by `TFloatField` to return a value of type `Double`. It returns the same value as TField.AsFloat (557)

See also: TField.AsFloat (557), TField.Value (566)

16.38.7 TFloatField.Currency

Synopsis: Is the field a currency field.

Declaration: `Property Currency : Boolean`

Visibility: published

Access: Read,Write

Description: `Currency` can be set to `True` to indicate that the field contains data representing an amount of currency. This affects the way the TField.DisplayText (563) and TField.Text (565) properties format the value of the field: if the `Currency` property is `True`, then these properties will format the value as a currency value (generally appending the currency sign) and if the `Currency` property is `False`, then they will format it as a normal floating-point value.

See also: TField.DisplayText (563), TField.Text (565), TNumericField.DisplayFormat (615), TNumericField.EditFormat (616)

16.38.8 TFloatField.MaxValue

Synopsis: Maximum value for the field.

Declaration: `Property MaxValue : Double`

Visibility: published

Access: Read,Write

Description: `MaxValue` can be set to a value different from zero, it is then the maximum value for the field if set to any value different from zero. When setting the field's value, the value may not be larger than `MaxValue`. Any attempt to write a larger value as the field's content will result in an exception. By default `MaxValue` equals `0`, i.e. any floating-point value is allowed.

If `MaxValue` is set, MinValue (592) should also be set, because it will also be checked.

See also: TFloatField.MinValue (592)

16.38.9 TFloatField.MinValue

Synopsis: Minimum value for the field.

Declaration: `Property MinValue : Double`

Visibility: published

Access: Read,Write

Description: `MinValue` can be set to a value different from zero, then it is the minimum value for the field. When setting the field's value, the value may not be less than `MinValue`. Any attempt to write a smaller value as the field's content will result in an exception. By default `MinValue` equals `0`, i.e. any floating-point value is allowed.

If `MinValue` is set, MaxValue (591) should also be set, because it will also be checked.

See also: TFloatField.MaxValue (591), TFloatField.CheckRange (591)

16.38.10 TFloatField.Precision

Synopsis: Precision (number of digits) of the field in text representations.

Declaration: `Property Precision : LongInt`

Visibility: published

Access: Read,Write

Description: `Precision` is the maximum number of digits that should be used when the field is converted to a textual representation in TField.Displaytext (563) or TField.Text (565), it is used in the arguments to FormatFloat (**??**).

See also: TField.Displaytext (563), TField.Text (565), FormatFloat (**??**)

16.39 TFMTBCDField

16.39.1 Description

`TFMTBCDField` is the field created when a data type of `ftFMTBCD` is encountered. It represents usually a fixed-precision floating point data type (BCD : Binary Coded Decimal data) such as the `DECIMAL` or `NUMERIC` field types in an SQL database.

See also: TFloatField (590)

16.39.2 Method overview

Page	Method	Description
593	CheckRange	Check value if it is in the range defined by MinValue and MaxValue.
593	Create	Create a new instance of the `TFMTBCDField` class.

16.39.3 Property overview

Page	Properties	Access	Description
594	Currency	rw	Does the field contain currency data ?
594	MaxValue	rw	Maximum value for the field.
594	MinValue	rw	Minimum value for the field.
594	Precision	rw	Total number of digits in the BCD data.
595	Size		Number of digits after the decimal point.
593	Value	rw	The value of the field as a BCD value.

16.39.4 TFMTBCDField.Create

Synopsis: Create a new instance of the `TFMTBCDField` class.

Declaration:
```
constructor Create(AOwner: TComponent);  Override
```

Visibility: public

Description: `Create` initializes a new instance of the `TFMTBCDField` class: it sets the MinValue (438), MaxValue (438), Size (565) (15) and Precision (438) (2) fields to their default values.

See also: MinValue (438), MaxValue (438), Size (565), Precision (438)

16.39.5 TFMTBCDField.CheckRange

Synopsis: Check value if it is in the range defined by MinValue and MaxValue.

Declaration:
```
function CheckRange(AValue: TBCD) : Boolean
```

Visibility: public

Description: `CheckRange` checks whether `AValue` is between MinValue (438) and MaxValue (438) if they are both nonzero. If either of them is zero, then `True` is returned. The `MinValue` and `MaxValue` values themselves are also valid values.

See also: MinValue (438), MaxValue (438)

16.39.6 TFMTBCDField.Value

Synopsis: The value of the field as a BCD value.

Declaration:
```
Property Value : TBCD
```

Visibility: public

Access: Read,Write

Description: `Value` is the value of the field as a BCD (Binary Coded Decimal) value.

See also: TField.AsFloat (557), TField.AsCurrency (556)

16.39.7 TFMTBCDField.Precision

Synopsis: Total number of digits in the BCD data.

Declaration: `Property Precision : LongInt`

Visibility: published

Access: Read,Write

Description: `Precision` is the total number of digits in the BCD data. The maximum precision is 32.

See also: TField.AsFloat (557), TField.AsCurrency (556), Size (438)

16.39.8 TFMTBCDField.Currency

Synopsis: Does the field contain currency data ?

Declaration: `Property Currency : Boolean`

Visibility: published

Access: Read,Write

Description: `Currency` determines how the textual representation of the data is formatted. It has no influence on the actual data itself. If `True` it is represented as a currency (monetary value). If DisplayFormat (549) or EditFormat (549) are set, these values are used instead to format the value.

See also: TField.DisplayFormat (549), TField.EditFormat (549)

16.39.9 TFMTBCDField.MaxValue

Synopsis: Maximum value for the field.

Declaration: `Property MaxValue : string`

Visibility: published

Access: Read,Write

Description: `MaxValue` can be set to a nonzero value to indicate the maximum value the field may contain. It must be set together with MinValue (438) or it will not have any effect.

See also: TFMTBCDField.CheckRange (593), MinValue (438)

16.39.10 TFMTBCDField.MinValue

Synopsis: Minimum value for the field.

Declaration: `Property MinValue : string`

Visibility: published

Access: Read,Write

Description: `MinValue` can be set to a nonzero value to indicate the maximum value the field may contain. It must be set together with MaxValue (438) or it will not have any effect.

See also: TFMTBCDField.CheckRange (593), MaxValue (438)

16.39.11 TFMTBCDField.Size

Synopsis: Number of digits after the decimal point.

Declaration: `Property Size :`

Visibility: published

Access:

Description: `Size` is the maximum number of digits allowed after the decimal point. Together with the Precision (438) property it determines the maximum allowed range of values for the field. This range can be restricted using the MinValue (438) and MaxValue (438) properties.

See also: MinValue (438), MaxValue (438), Precision (438)

16.40 TGraphicField

16.40.1 Description

`TGraphicField` is the class used when a dataset must manage graphical BLOB data. (TField.DataType (562) equals `ftGraphic`). It initializes some of the properties of the TField (549) class. All methods to be able to work with graphical BLOB data have been implemented in the TBlobField (471) parent class.

It should never be necessary to create an instance of `TGraphicsField` manually, a field of this class will be instantiated automatically for each graphical BLOB field when a dataset is opened.

See also: TDataset (496), TField (549), TBLOBField (471), TMemoField (612), TWideMemoField (645)

16.40.2 Method overview

Page	Method	Description
595	Create	Create a new instance of the `TGraphicField` class.

16.40.3 TGraphicField.Create

Synopsis: Create a new instance of the `TGraphicField` class.

Declaration: `constructor Create(AOwner: TComponent); Override`

Visibility: public

Description: `Create` initializes a new instance of the `TGraphicField` class. It calls the inherited destructor, and then sets some TField (549) properties to configure the instance for working with graphical BLOB values.

See also: TField (549)

16.41 TGuidField

16.41.1 Description

`TGUIDField` is the class used when a dataset must manage native variant-typed data. (TField.DataType (562) equals `ftGUID`). It initializes some of the properties of the TField (549) class and overrides

some of its methods to be able to work with variant data. It also adds a method to retrieve the field value as a native `TGUID` type.

It should never be necessary to create an instance of `TGUIDField` manually, a field of this class will be instantiated automatically for each GUID field when a dataset is opened.

See also: TDataset (496), TField (549), TGuidField.AsGuid (596)

16.41.2 Method overview

Page	Method	Description
596	Create	Create a new instance of the `TGUIDField` class.

16.41.3 Property overview

Page	Properties	Access	Description
596	AsGuid	rw	Field content as a GUID value.

16.41.4 TGuidField.Create

Synopsis: Create a new instance of the `TGUIDField` class.

Declaration:
```
constructor Create(AOwner: TComponent);  Override
```

Visibility: public

Description: `Create` initializes a new instance of the `TGUIDField` class. It calls the inherited destructor, and then sets some TField (549) properties to configure the instance for working with GUID values.

See also: TField (549)

16.41.5 TGuidField.AsGuid

Synopsis: Field content as a GUID value.

Declaration:
```
Property AsGuid : TGUID
```

Visibility: public

Access: Read,Write

Description: `AsGUID` can be used to get or set the field's content as a value of type `TGUID`.

See also: TField.AsString (559)

16.42 TIndexDef

16.42.1 Description

`TIndexDef` describes one index in a set of indexes of a TDataset (496) instance. The collection of indexes is described by the TIndexDefs (599) class. It just has the necessary properties to describe an index, but does not implement any functionality to maintain an index.

See also: TIndexDefs (599)

16.42.2 Method overview

Page	Method	Description
597	Create	Create a new index definition.

16.42.3 Property overview

Page	Properties	Access	Description
598	CaseInsFields	rw	Fields in field list that are ordered case-insensitively.
598	DescFields	rw	Fields in field list that are ordered descending.
597	Expression	rw	Expression that makes up the index values.
597	Fields	rw	Fields making up the index.
598	Options	rw	Index options.
599	Source	rw	Source of the index.

16.42.4 TIndexDef.Create

Synopsis: Create a new index definition.

Declaration:
```
constructor Create(Owner: TIndexDefs; const AName: string;
                   const TheFields: string; TheOptions: TIndexOptions)
                   ;  Overload
```

Visibility: public

Description: `Create` initializes a new TIndexDef (596) instance with the `AName` value as the index name, `AField` as the fields making up the index, and `TheOptions` as the options. `Owner` should be the TIndexDefs (599) instance to which the new `TIndexDef` can be added.

Errors: If an index with name `AName` already exists in the collection, an exception will be raised.

See also: TIndexDefs (599), TIndexDef.Options (598), TIndexDef.Fields (597)

16.42.5 TIndexDef.Expression

Synopsis: Expression that makes up the index values.

Declaration:
```
Property Expression : string
```

Visibility: published

Access: Read,Write

Description: `Expression` is an SQL expression based on which the index values are computed. It is only used when `ixExpression` is in TIndexDef.Options (598)

See also: TIndexDef.Options (598), TindexDef.Fields (597)

16.42.6 TIndexDef.Fields

Synopsis: Fields making up the index.

Declaration:
```
Property Fields : string
```

Visibility: published

Access: Read,Write

Description: `Fields` is a list of fieldnames, separated by semicolons: the fields that make up the index, in case the index is not based on an expression. The list contains the names of all fields, regardless of whether the sort order for a particular field is ascending or descending. The fields should be in the right order, i.e. the first field is sorted on first, and so on.

The TIndexDef.DescFields (598) property can be used to determine the fields in the list that have a descending sort order. The TIndexDef.CaseInsFields (598) property determines which fields are sorted in a case-insensitive manner.

See also: TIndexDef.DescFields (598), TIndexDef.CaseInsFields (598), TIndexDef.Expression (597)

16.42.7 TIndexDef.CaseInsFields

Synopsis: Fields in field list that are ordered case-insensitively.

Declaration: `Property CaseInsFields : string`

Visibility: published

Access: Read,Write

Description: `CaseInsFields` is a list of fieldnames, separated by semicolons. It contains the names of the fields in the Fields (597) property which are ordered in a case-insensitive manner. `CaseInsFields` may not contain fieldnames that do not appear in `Fields`.

See also: TIndexDef.Fields (597), TIndexDef.Expression (597), TIndexDef.DescFields (598)

16.42.8 TIndexDef.DescFields

Synopsis: Fields in field list that are ordered descending.

Declaration: `Property DescFields : string`

Visibility: published

Access: Read,Write

Description: `DescFields` is a list of fieldnames, separated by semicolons. It contains the names of the fields in the Fields (597) property which are ordered in a descending manner. `DescFields` may not contain fieldnames that do not appear in `Fields`.

See also: TIndexDef.Fields (597), TIndexDef.Expression (597), TIndexDef.DescFields (598)

16.42.9 TIndexDef.Options

Synopsis: Index options.

Declaration: `Property Options : TIndexOptions`

Visibility: published

Access: Read,Write

Description: `Options` describes the various properties of the index. This is usually filled by the dataset that provides the index definitions. For datasets that provide In-memory indexes, this should be set prior to creating the index: it cannot be changed once the index is created.

See the description of TindexOption (450) for more information on the various available options.

See also: TIndexOptions (450)

16.42.10 TIndexDef.Source

Synopsis: Source of the index.

Declaration: `Property Source : string`

Visibility: published

Access: Read,Write

Description: `Source` describes where the index comes from. This is a property for the convenience of the various datasets that provide indexes: they can use it to describe the source of the index.

16.43 TIndexDefs

16.43.1 Description

`TIndexDefs` is used to keep a collection of index (sort order) definitions. It can be used by classes that provide in-memory or on-disk indexes to provide a list of available indexes.

See also: TIndexDef (596), TIndexDefs.Items (601)

16.43.2 Method overview

Page	Method	Description
600	Add	Add a new index definition with given name and options.
600	AddIndexDef	Add a new, empty, index definition.
599	Create	Create a new `TIndexDefs` instance.
600	Find	Find an index by name.
600	FindIndexForFields	Find index definition based on field names.
601	GetIndexForFields	Get index definition based on field names.
601	Update	Called whenever one of the items changes.

16.43.3 Property overview

Page	Properties	Access	Description
601	Items	rw	Indexed access to the index definitions.

16.43.4 TIndexDefs.Create

Synopsis: Create a new `TIndexDefs` instance.

Declaration: `constructor Create(ADataSet: TDataSet); Virtual; Overload`

Visibility: public

Description: `Create` initializes a new instance of the `TIndexDefs` class. It simply calls the inherited destructor with the appropriate item class, TIndexDef (596).

See also: TIndexDef (596), TIndexDefs.Destroy (599)

16.43.5 TIndexDefs.Add

Synopsis: Add a new index definition with given name and options.

Declaration:
```
procedure Add(const Name: string; const Fields: string;
                Options: TIndexOptions);  Overload
```

Visibility: public

Description: `Add` adds a new TIndexDef (596) instance to the list of indexes. It initializes the index definition properties `Name`, `Fields` and `Options` with the values given in the parameters with the same names.

Errors: If an index with the same `Name` already exists in the list of indexes, an exception will be raised.

See also: TIndexDef (596), TNamedItem.Name (614), TIndexDef.Fields (597), TIndexDef.Options (598), TIndexDefs.AddIndexDef (600)

16.43.6 TIndexDefs.AddIndexDef

Synopsis: Add a new, empty, index definition.

Declaration:
```
function AddIndexDef : TIndexDef
```

Visibility: public

Description: `AddIndexDef` adds a new TIndexDef (596) instance to the list of indexes, and returns the newly created instance. It does not initialize any of the properties of the new index definition.

See also: TIndexDefs.Add (600)

16.43.7 TIndexDefs.Find

Synopsis: Find an index by name.

Declaration:
```
function Find(const IndexName: string) : TIndexDef
```

Visibility: public

Description: `Find` overloads the TDefCollection.Find (546) method to search and return a TIndexDef (596) instance based on the name. The search is case-insensitive and raises an exception if no matching index definition was found. Note: `TIndexDefs.IndexOf` can be used instead if an exception is not desired.

See also: TIndexDef (596), TDefCollection.Find (546), TIndexDefs.FindIndexForFields (600)

16.43.8 TIndexDefs.FindIndexForFields

Synopsis: Find index definition based on field names.

Declaration:
```
function FindIndexForFields(const Fields: string) : TIndexDef
```

Visibility: public

Description: `FindIndexForFields` searches in the list of indexes for an index whose TIndexDef.Fields (597) property matches the list of fields in `Fields`. If it finds an index definition, then it returns the found instance.

Errors: If no matching definition is found, an exception is raised. This is different from other Find functionality, where Find usually returns Nil if nothing is found.

See also: TIndexDef (596), TIndexDefs.Find (600), TIndexDefs.GetindexForFields (601)

16.43.9 TIndexDefs.GetIndexForFields

Synopsis: Get index definition based on field names.

Declaration:
```
function GetIndexForFields(const Fields: string;
                           CaseInsensitive: Boolean) : TIndexDef
```

Visibility: public

Description: `GetIndexForFields` searches in the list of indexes for an index whose TIndexDef.Fields (597) property matches the list of fields in `Fields`. If `CaseInsenstitive` is `True` it only searches for case-sensitive indexes. If it finds an index definition, then it returns the found instance. If it does not find a matching definition, `Nil` is returned.

See also: TIndexDef (596), TIndexDefs.Find (600), TIndexDefs.FindIndexForFields (600)

16.43.10 TIndexDefs.Update

Synopsis: Called whenever one of the items changes.

Declaration:
```
procedure Update;  Virtual;  Overload
```

Visibility: public

Description: `Update` can be called to have the dataset update its index definitions.

16.43.11 TIndexDefs.Items

Synopsis: Indexed access to the index definitions.

Declaration:
```
Property Items[Index: Integer]: TIndexDef; default
```

Visibility: public

Access: Read,Write

Description: `Items` is redefined by `TIndexDefs` using `TIndexDef` as the type for the elements. It is the default property of the `TIndexDefs` class.

See also: TIndexDef (596)

16.44 TIntegerField

16.44.1 Description

`TIntegerField` is an alias for TLongintField (603).

See also: TLongintField (603), TField (549)

16.45 TLargeintField

16.45.1 Description

`TLargeIntField` is instantiated when a dataset must manage a field with 64-bit signed data: the data type `ftLargeInt`. It overrides some methods of TField (549) to handle int64 data, and

sets some of the properties to values for int64 data. It also introduces some methods and properties specific to 64-bit integer data such as MinValue (603) and MaxValue (603).

It should never be necessary to create an instance of `TLargeIntField` manually, a field of this class will be instantiated automatically for each int64 field when a dataset is opened.

See also: TField (549), MinValue (603), MaxValue (603)

16.45.2 Method overview

Page	Method	Description
602	CheckRange	Check whether a values falls within the allowed range.
602	Create	Create a new instance of the `TLargeintField` class.

16.45.3 Property overview

Page	Properties	Access	Description
603	MaxValue	rw	Maximum value for the field.
603	MinValue	rw	Minimum value for the field.
602	Value	rw	Field contents as a 64-bit integer value.

16.45.4 TLargeintField.Create

Synopsis: Create a new instance of the `TLargeintField` class.

Declaration:
```
constructor Create(AOwner: TComponent);  Override
```

Visibility: public

Description: `Create` initializes a new instance of the `TLargeIntField` class: it calls the inherited constructor and then initializes the various properties of Tfield (549) and MinValue (603) and MaxValue (603).

See also: TField (549), MinValue (603), MaxValue (603)

16.45.5 TLargeintField.CheckRange

Synopsis: Check whether a values falls within the allowed range.

Declaration:
```
function CheckRange(AValue: LargeInt) : Boolean
```

Visibility: public

Description: `CheckRange` returns `True` if `AValue` lies within the range defined by the MinValue (603) and MaxValue (603) properties. If the value lies outside of the allowed range, then `False` is returned.

See also: MaxValue (603), MinValue (603)

16.45.6 TLargeintField.Value

Synopsis: Field contents as a 64-bit integer value.

Declaration:
```
Property Value : LargeInt
```

Visibility: public

Access: Read,Write

Description: `Value` is redefined by `TLargeIntField` as a 64-bit integer value. It returns the same value as TField.AsLargeInt (558).

See also: TField.Value (566), TField.AsLargeInt (558)

16.45.7 TLargeintField.MaxValue

Synopsis: Maximum value for the field.

Declaration: `Property MaxValue : LargeInt`

Visibility: published

Access: Read,Write

Description: `MaxValue` is the maximum value for the field if set to any value different from zero. When setting the field's value, the value may not be larger than `MaxValue`. Any attempt to write a larger value as the field's content will result in an exception. By default `MaxValue` equals `0`, i.e. any integer value is allowed.

If `MaxValue` is set, MinValue (603) should also be set, because it will also be checked.

See also: TLargeIntField.MinValue (603)

16.45.8 TLargeintField.MinValue

Synopsis: Minimum value for the field.

Declaration: `Property MinValue : LargeInt`

Visibility: published

Access: Read,Write

Description: `MinValue` is the minimum value for the field. When setting the field's value, the value may not be less than `MinValue`. Any attempt to write a smaller value as the field's content will result in an exception. By default `MinValue` equals `0`, i.e. any integer value is allowed.

If `MinValue` is set, MaxValue (603) should also be set, because it will also be checked.

See also: TLargeIntField.MaxValue (603)

16.46 TLongintField

16.46.1 Description

`TLongintField` is instantiated when a dataset must manage a field with 32-bit signed data: the data type `ftInteger`. It overrides some methods of TField (549) to handle integer data, and sets some of the properties to values for integer data. It also introduces some methods and properties specific to integer data such as MinValue (605) and MaxValue (605).

It should never be necessary to create an instance of `TLongintField` manually, a field of this class will be instantiated automatically for each integer field when a dataset is opened.

See also: TField (549), MaxValue (605), MinValue (605)

16.46.2 Method overview

Page	Method	Description
604	CheckRange	Check whether a valid is in the allowed range of values for the field.
604	Create	Create a new instance of `TLongintField`.

16.46.3 Property overview

Page	Properties	Access	Description
605	MaxValue	rw	Maximum value for the field.
605	MinValue	rw	Minimum value for the field.
604	Value	rw	Value of the field as longint.

16.46.4 TLongintField.Create

Synopsis: Create a new instance of `TLongintField`.

Declaration:
```
constructor Create(AOwner: TComponent);  Override
```

Visibility: public

Description: `Create` initializes a new instance of `TLongintField`. After calling the inherited constructor, it initializes the MinValue (605) and MaxValue (605) properties.

See also: TField (549), MaxValue (605), MinValue (605)

16.46.5 TLongintField.CheckRange

Synopsis: Check whether a valid is in the allowed range of values for the field.

Declaration:
```
function CheckRange(AValue: LongInt) : Boolean
```

Visibility: public

Description: `CheckRange` returns `True` if `AValue` lies within the range defined by the MinValue (605) and MaxValue (605) properties. If the value lies outside of the allowed range, then `False` is returned.

See also: MaxValue (605), MinValue (605)

16.46.6 TLongintField.Value

Synopsis: Value of the field as longint.

Declaration:
```
Property Value : LongInt
```

Visibility: public

Access: Read,Write

Description: `Value` is redefined by `TLongintField` as a 32-bit signed integer value. It returns the same value as the TField.AsInteger (558) property.

See also: TField.Value (566)

16.46.7 TLongintField.MaxValue

Synopsis: Maximum value for the field.

Declaration: `Property MaxValue : LongInt`

Visibility: published

Access: Read,Write

Description: `MaxValue` is the maximum value for the field. When setting the field's value, the value may not be larger than `MaxValue`. Any attempt to write a larger value as the field's content will result in an exception. By default `MaxValue` equals `MaxInt`, i.e. any integer value is allowed.

See also: MinValue (605)

16.46.8 TLongintField.MinValue

Synopsis: Minimum value for the field.

Declaration: `Property MinValue : LongInt`

Visibility: published

Access: Read,Write

Description: `MinValue` is the minimum value for the field. When setting the field's value, the value may not be less than `MinValue`. Any attempt to write a smaller value as the field's content will result in an exception. By default `MinValue` equals `-MaxInt`, i.e. any integer value is allowed.

See also: MaxValue (605)

16.47 TLongWordField

16.47.1 Description

`TByteField` is instantiated when a dataset must manage a field with 32-bit unsigned data: the data type `ftLongword`. It overrides some methods of TField (549) to handle `LongWord` data, and sets some of the properties to values for `LongWord` data. It also introduces some methods and properties specific to integer data such as MinValue (607) and MaxValue (606).

It should never be necessary to create an instance of `TLongWordField` manually, a field of this class will be instantiated automatically for each integer field when a dataset is opened.

See also: MinValue (607), MaxValue (606)

16.47.2 Method overview

Page	Method	Description
606	CheckRange	Checkif a value is in the allowed range.
606	Create	Create new instance of `TLongWordField`.

16.47.3 Property overview

Page	Properties	Access	Description
606	MaxValue	rw	Maximum field value.
607	MinValue	rw	Minimum field value.
606	Value	rw	Value as longword.

16.47.4 TLongWordField.Create

Synopsis: Create new instance of `TLongWordField`.

Declaration: `constructor Create(AOwner: TComponent); Override`

Visibility: public

Description: `Create` calls the inherited constructor and sets the values of the MinValue (607)MaxValue (606) and TField.DataType (562) properties.

See also: MinValue (605), MaxValue (605), TField.DataType (562)

16.47.5 TLongWordField.CheckRange

Synopsis: Checkif a value is in the allowed range.

Declaration: `function CheckRange(AValue: LargeInt) : Boolean`

Visibility: public

Description: `CheckRange` checks whether `aValue` is in the range of allowed values. This is normally `[0..High(LongWord)]`, unless specified otherwise in MinValue (607) or MaxValue (606)

See also: MinValue (607), MaxValue (606)

16.47.6 TLongWordField.Value

Synopsis: Value as longword.

Declaration: `Property Value : LongWord`

Visibility: public

Access: Read,Write

Description: `Value` is reintroduced in `TLongWordField` and gives access to the value of the field as `LongWord` data (a `Cardinal`).

See also: TField.Value (566)

16.47.7 TLongWordField.MaxValue

Synopsis: Maximum field value.

Declaration: `Property MaxValue : LongWord`

Visibility: published

Access: Read,Write

Description: `MaxValue` is the maximum value the field contents can have. It is checked when setting the field value. By default it is set to `High(LongWord)` but you can set it to a smaller value to limit the range of allowed values.

See also: MinValue (607), CheckRange (606)

16.47.8 TLongWordField.MinValue

Synopsis: Minimum field value.

Declaration: `Property MinValue : LongWord`

Visibility: published

Access: Read,Write

Description: `MinValue` is the minimum value the field contents can have. It is checked when setting the field value. By default it is set to `Low(LongWord)` (Zero) but you can set it to a larger value to limit the range of allowed values.

See also: MaxValue (606), CheckRange (606)

16.48 TLookupList

16.48.1 Description

`TLookupList` is a list object used for storing values of lookup operations by lookup fields. There should be no need to create an instance of `TLookupList` manually, the `TField` instance will create an instance of `TlookupList` on demand.

See also: TField.LookupCache (571)

16.48.2 Method overview

Page	Method	Description
608	Add	Add a key, value pair to the list.
608	Clear	Remove all key, value pairs from the list.
607	Create	Create a new instance of `TLookupList`.
607	Destroy	Free a `TLookupList` instance from memory.
608	FirstKeyByValue	Find the first key that matches a value.
608	ValueOfKey	Look up value based on a key.
609	ValuesToStrings	Convert values to stringlist.

16.48.3 TLookupList.Create

Synopsis: Create a new instance of `TLookupList`.

Declaration: `constructor Create`

Visibility: public

Description: `Create` sets up the necessary structures to manage a list of lookup values for a lookup field.

See also: TLookupList.Destroy (607)

16.48.4 TLookupList.Destroy

Synopsis: Free a `TLookupList` instance from memory.

Declaration: `destructor Destroy;  Override`

Visibility: public

Description: `Destroy` frees all resources (mostly memory) allocated by the lookup list, and calls then the inherited destructor.

See also: TLookupList.Create (607)

16.48.5 TLookupList.Add

Synopsis: Add a key, value pair to the list.

Declaration: `procedure Add(const AKey: Variant; const AValue: Variant)`

Visibility: public

Description: `Add` will add the value `AValue` to the list and associate it with key `AKey`. The same key cannot be added twice.

See also: TLookupList.Clear (608)

16.48.6 TLookupList.Clear

Synopsis: Remove all key, value pairs from the list.

Declaration: `procedure Clear`

Visibility: public

Description: `Clear` removes all keys and associated values from the list.

See also: TLookupList.Add (608)

16.48.7 TLookupList.FirstKeyByValue

Synopsis: Find the first key that matches a value.

Declaration: `function FirstKeyByValue(const AValue: Variant) : Variant`

Visibility: public

Description: `FirstKeyByValue` does a reverse lookup: it returns the first key value in the list that matches the `AValue` value. If none is found, `Null` is returned. This mechanism is quite slow, as a linear search is performed.

Errors: If no key is found, `Null` is returned.

See also: TLookupList.ValueOfKey (608)

16.48.8 TLookupList.ValueOfKey

Synopsis: Look up value based on a key.

Declaration: `function ValueOfKey(const AKey: Variant) : Variant`

Visibility: public

Description: `ValueOfKey` does a value lookup based on a key: it returns the value in the list that matches the `AKey` key. If none is found, `Null` is returned. This mechanism is quite slow, as a linear search is performed.

See also: TLookupList.FirstKeyByValue (608), TLookupList.Add (608)

16.48.9 TLookupList.ValuesToStrings

Synopsis: Convert values to stringlist.

Declaration: `procedure ValuesToStrings(AStrings: TStrings)`

Visibility: public

Description: `ValuesToStrings` converts the list of values to a stringlist, so they can be used e.g. in a drop-down list.

See also: TLookupList.ValueOfKey (608)

16.49 TMasterDataLink

16.49.1 Description

`TMasterDataLink` is a `TDatalink` descendent which handles master-detail relations. It can be used in TDataset (496) descendents that must have master-detail functionality: the detail dataset creates an instance of `TMasterDataLink` to point to the master dataset, which is subsequently available through the TDataLink.Dataset (495) property.

The class also provides functionality for keeping a list of fields that make up the master-detail functionality, in the TMasterDatalink.FieldNames (610) and TMasterDataLink.Fields (610) properties.

This class should never be used in application code.

See also: TDataset (496), TDatalink.DataSource (495), TDatalink.DataSet (495), TMasterDatalink.FieldNames (610), TMasterDataLink.Fields (610)

16.49.2 Method overview

Page	Method	Description
609	Create	Create a new instance of `TMasterDataLink`.
610	Destroy	Free the datalink instance from memory.

16.49.3 Property overview

Page	Properties	Access	Description
610	FieldNames	rw	List of fieldnames that make up the master-detail relationship.
610	Fields	r	List of fields as specified in `FieldNames`.
610	OnMasterChange	rw	Called whenever the master dataset data changes.
611	OnMasterDisable	rw	Called whenever the master dataset is disabled.

16.49.4 TMasterDataLink.Create

Synopsis: Create a new instance of `TMasterDataLink`.

Declaration: `constructor Create(ADataSet: TDataSet); Virtual`

Visibility: public

Description: `Create` initializes a new instance of `TMasterDataLink`. The `ADataset` parameter is the detail dataset in the master-detail relation: it is saved in the DetailDataset (547) property. The master dataset must be set through the DataSource (495) property, and is usually set by the application programmer.

See also: TDetailDataLink.DetailDataset (547), TDatalink.Datasource (495)

16.49.5 TMasterDataLink.Destroy

Synopsis: Free the datalink instance from memory.

Declaration: `destructor Destroy;  Override`

Visibility: public

Description: `Destroy` cleans up the resources used by `TMasterDatalink` and then calls the inherited destructor.

See also: TMasterDatalink.Create (609)

16.49.6 TMasterDataLink.FieldNames

Synopsis: List of fieldnames that make up the master-detail relationship.

Declaration: `Property FieldNames : string`

Visibility: public

Access: Read,Write

Description: `FieldNames` is a semicolon-separated list of fieldnames in the master dataset (TDatalink.Dataset (495)) on which the master-detail relationship is based. Setting this property will fill the TMasterDataLink.Fields (610) property with the field instances of the master dataset.

See also: TMasterDataLink.Fields (610), TDatalink.Dataset (495), TDataset.GetFieldList (511)

16.49.7 TMasterDataLink.Fields

Synopsis: List of fields as specified in `FieldNames`.

Declaration: `Property Fields : TList`

Visibility: public

Access: Read

Description: `Fields` is filled with the TField (549) instances from the master dataset (TDatalink.Dataset (495)) when the FieldNames (610) property is set, and when the master dataset opens.

See also: TField (549), TMasterDatalink.FieldNames (610)

16.49.8 TMasterDataLink.OnMasterChange

Synopsis: Called whenever the master dataset data changes.

Declaration: `Property OnMasterChange : TNotifyEvent`

Visibility: public

Access: Read,Write

Description: `OnMasterChange` is called whenever the field values in the master dataset changes, i.e. when it becomes active, or when the current record changes. If the TMasterDataLink.Fields (610) list is empty, TMasterDataLink.OnMasterDisable (611) is called instead.

See also: TMasterDataLink.OnMasterDisable (611)

16.49.9 TMasterDataLink.OnMasterDisable

Synopsis: Called whenever the master dataset is disabled.

Declaration: `Property OnMasterDisable : TNotifyEvent`

Visibility: public

Access: Read,Write

Description: `OnMasterDisable` is called whenever the master dataset is disabled, or when it is active and the field list is empty.

See also: TMasterDataLink.OnMasterChange (610)

16.50 TMasterParamsDataLink

16.50.1 Description

`TMasterParamsDataLink` is a TDataLink (491) descendent that can be used to establish a master-detail relationship between 2 `TDataset` instances where the detail dataset is parameterized using a `TParams` instance. It takes care of closing and opening the detail dataset and copying the parameter values from the master dataset whenever the data in the master dataset changes.

See also: TDatalink (491), TDataset (496), TParams (631), TParam (617)

16.50.2 Method overview

Page	Method	Description
612	CopyParamsFromMaster	Copy parameter values from master dataset.
611	Create	Initialize a new `TMasterParamsDataLink` instance.
612	RefreshParamNames	Refresh the list of parameter names.

16.50.3 Property overview

Page	Properties	Access	Description
612	Params	rw	Parameters of detail dataset.

16.50.4 TMasterParamsDataLink.Create

Synopsis: Initialize a new `TMasterParamsDataLink` instance.

Declaration: `constructor Create(ADataSet: TDataSet); Override`

Visibility: public

Description: `Create` first calls the inherited constructor using `ADataset`, and then looks for a property named `Params` of type TParams (631) in the published properties of `ADataset` and assigns it to the Params (612) property.

See also: TDataset (496), TParams (631), TMasterParamsDataLink.Params (612)

16.50.5 TMasterParamsDataLink.RefreshParamNames

Synopsis: Refresh the list of parameter names.

Declaration: `procedure RefreshParamNames;  Virtual`

Visibility: public

Description: `RefreshParamNames` scans the Params (612) property and sets the FieldNames (610) property to the list of parameter names.

See also: TMasterParamsDataLink.Params (612), TMasterDataLink.FieldNames (610)

16.50.6 TMasterParamsDataLink.CopyParamsFromMaster

Synopsis: Copy parameter values from master dataset.

Declaration: `procedure CopyParamsFromMaster(CopyBound: Boolean);  Virtual`

Visibility: public

Description: `CopyParamsFromMaster` calls TParams.CopyParamValuesFromDataset (636), passing it the master dataset: it provides the parameters of the detail dataset with their new values. If `CopyBound` is `false`, then only parameters with their Bound (628) property set to `False` are copied. If it is `True` then the value is set for all parameters.

Errors: If the master dataset does not have a corresponding field for each parameter, then an exception will be raised.

See also: TParams.CopyParamValuesFromDataset (636), TParam.Bound (628)

16.50.7 TMasterParamsDataLink.Params

Synopsis: Parameters of detail dataset.

Declaration: `Property Params : TParams`

Visibility: public

Access: Read,Write

Description: `Params` is the `TParams` instance of the detail dataset. If the detail dataset contains a property named `Params` of type `TParams`, then it will be set when the `TMasterParamsDataLink` instance was created. If the property is not published, or has another name, then the `Params` property must be set in code.

See also: Tparams (631), TMasterParamsDataLink.Create (611)

16.51 TMemoField

16.51.1 Description

`TMemoField` is the class used when a dataset must manage memo (Text BLOB) data. (TField.DataType (562) equals `ftMemo`). It initializes some of the properties of the TField (549) class. All methods to be able to work with memo fields have been implemented in the TBlobField (471) parent class.

It should never be necessary to create an instance of `TMemoField` manually, a field of this class will be instantiated automatically for each memo field when a dataset is opened.

See also: TDataset (496), TField (549), TBLOBField (471), TWideMemoField (645), TGraphicField (595)

16.51.2 Method overview

Page	Method	Description
613	Create	Create a new instance of the `TMemoField` class.

16.51.3 Property overview

Page	Properties	Access	Description
613	CodePage	r	Codepage of the memo field string data.
613	Transliterate		Should the contents of the field be transliterated.

16.51.4 TMemoField.Create

Synopsis: Create a new instance of the `TMemoField` class.

Declaration:
```
constructor Create(AOwner: TComponent);  Override
```

Visibility: public

Description: `Create` initializes a new instance of the `TMemoField` class. It calls the inherited destructor, and then sets some TField (549) properties to configure the instance for working with memo values.

See also: TField (549)

16.51.5 TMemoField.CodePage

Synopsis: Codepage of the memo field string data.

Declaration:
```
Property CodePage : TSystemCodePage
```

Visibility: public

Access: Read

Description: `CodePage` is the code page of the string data in the field. It is determined when the field is initially created from the dataset's data, and cannot be changed while the dataset is active.

See also: TField.AsString (559), TFieldDef.CodePage (579)

16.51.6 TMemoField.Transliterate

Synopsis: Should the contents of the field be transliterated.

Declaration:
```
Property Transliterate :
```

Visibility: published

Access:

Description: `Transliterate` is redefined from TBlobField.Transliterate (475) with a default value of `true`.

See also: TBlobField.Transliterate (475), TStringField.Transliterate (642), TDataset.Translate (517)

16.52 TNamedItem

16.52.1 Description

`NamedItem` is a TCollectionItem (**??**) descendent which introduces a Name (614) property. It automatically returns the value of the `Name` property as the value of the DisplayName (614) property.

See also: DisplayName (614), Name (614)

16.52.2 Property overview

Page	Properties	Access	Description
614	DisplayName	rw	Display name.
614	Name	rw	Name of the item.

16.52.3 TNamedItem.DisplayName

Synopsis: Display name.

Declaration: `Property DisplayName : string`

Visibility: public

Access: Read,Write

Description: `DisplayName` is declared in TCollectionItem (**??**), and is made public in `TNamedItem`. The value equals the value of the Name (614) property.

See also: Name (614)

16.52.4 TNamedItem.Name

Synopsis: Name of the item.

Declaration: `Property Name : string`

Visibility: published

Access: Read,Write

Description: `Name` is the name of the item in the collection. This property is also used as the value for the DisplayName (614) property. If the `TNamedItem` item is owned by a TDefCollection (545) collection, then the name must be unique, i.e. each `Name` value may appear only once in the collection.

See also: DisplayName (614), TDefCollection (545)

16.53 TNumericField

16.53.1 Description

`TNumericField` is an abstract class which overrides some of the methods of TField (549) to handle numerical data. It also introduces or publishes a couple of properties that are only relevant in the case of numerical data, such as TNumericField.DisplayFormat (615) and TNumericField.EditFormat (616).

Since `TNumericField` is an abstract class, it must never be instantiated directly. Instead one of the descendent classes should be created.

See also: TField (549), TNumericField.DisplayFormat (615), TNumericField.EditFormat (616), TField.Alignment (567), TIntegerField (601), TLargeIntField (601), TFloatField (590), TBCDField (467)

16.53.2 Method overview

Page	Method	Description
615	Create	Create a new instance of `TNumericField`.

16.53.3 Property overview

Page	Properties	Access	Description
615	Alignment		Alignment of the field.
615	DisplayFormat	rw	Format string for display of numerical data.
616	EditFormat	rw	Format string for editing of numerical data.

16.53.4 TNumericField.Create

Synopsis: Create a new instance of `TNumericField`.

Declaration: `constructor Create(AOwner: TComponent); Override`

Visibility: public

Description: `Create` calls the inherited constructor and then initializes the TField.Alignment (567) property with

See also: TField.Alignment (567)

16.53.5 TNumericField.Alignment

Synopsis: Alignment of the field.

Declaration: `Property Alignment :`

Visibility: published

Access:

Description: `Alignment` is published by `TNumericField` with `taRightJustify` as a default value.

See also: TField.Alignment (567)

16.53.6 TNumericField.DisplayFormat

Synopsis: Format string for display of numerical data.

Declaration: `Property DisplayFormat : string`

Visibility: published

Access: Read,Write

Description: `DisplayFormat` specifies a format string (such as used by the Format (**??**) and FormatFloat (**??**) functions) for display purposes: the TField.DisplayText (563) property will use this property to format the field's value. Which formatting function (and, consequently, which format can be entered) is used depends on the descendent of the `TNumericField` class.

See also: Format (**??**), FormatFloat (**??**), TField.DisplayText (563), TNumericField.EditFormat (616)

16.53.7 TNumericField.EditFormat

Synopsis: Format string for editing of numerical data.

Declaration: `Property EditFormat : string`

Visibility: published

Access: Read,Write

Description: `EditFormat` specifies a format string (such as used by the Format (**??**) and FormatFloat (**??**) functions) for editing purposes: the TField.Text (565) property will use this property to format the field's value. Which formatting function (and, consequently, which format can be entered) is used depends on the descendent of the `TNumericField` class.

See also: Format (**??**), FormatFloat (**??**), TField.Text (565), TNumericField.DisplayFormat (615)

16.54 TObjectField

16.54.1 Property overview

Page	Properties	Access	Description
616	FieldCount	r	
616	Fields	r	
616	FieldValues	rw	
617	ObjectType	rw	
617	UnNamed	r	

16.54.2 TObjectField.FieldCount

Declaration: `Property FieldCount : Integer`

Visibility: public

Access: Read

16.54.3 TObjectField.Fields

Declaration: `Property Fields : TFields`

Visibility: public

Access: Read

16.54.4 TObjectField.FieldValues

Declaration: `Property FieldValues[AIndex: Integer]: Variant; default`

Visibility: public

Access: Read,Write

16.54.5 TObjectField.UnNamed

Declaration: `Property UnNamed : Boolean`

Visibility: public

Access: Read

16.54.6 TObjectField.ObjectType

Declaration: `Property ObjectType : string`

Visibility: published

Access: Read,Write

16.55 TParam

16.55.1 Description

`TParam` is one item in a TParams (631) collection. It describes the name (TParam.Name (629)), type (ParamType (630)) and value (TParam.Value (631)) of a parameter in a parameterized query or stored procedure. Under normal circumstances, it should never be necessary to create a `TParam` instance manually; the TDataset (496) descendent that owns the parameters should have created all necessary `TParam` instances.

See also: TParams (631)

16.55.2 Method overview

Page	Method	Description
619	Assign	Assign one parameter instance to another.
619	AssignField	Copy value from field instance.
619	AssignFieldValue	Assign field value to the parameter.
620	AssignFromField	Copy field type and value.
619	AssignToField	Assign parameter value to field.
620	Clear	Clear the parameter value.
618	Create	Create a new parameter value.
620	GetData	Get the parameter value from a memory buffer.
620	GetDataSize	Return the size of the data.
621	LoadFromFile	Load a parameter value from file.
621	LoadFromStream	Load a parameter value from stream.
621	SetBlobData	Set BLOB data.
621	SetData	Set the parameter value from a buffer.

16.55.3 Property overview

Page	Properties	Access	Description
626	AsAnsiString	rw	Parameter contents as an ANSI string.
622	AsBCD	rw	Get or set parameter value as BCD value.
622	AsBlob	rw	Return parameter value as a blob.
622	AsBoolean	rw	Get/Set parameter value as a boolean value.
623	AsByte	rw	Get/Set parameter value as a 8-bit unsigned integer value.
623	AsBytes	rw	Get or set parameter value as `TBytes`.
623	AsCurrency	rw	Get/Set parameter value as a currency value.
623	AsDate	rw	Get/Set parameter value as a date (TDateTime) value.
624	AsDateTime	rw	Get/Set parameter value as a date/time (TDateTime) value.
624	AsFloat	rw	Get/Set parameter value as a floating-point value.
627	AsFMTBCD	rw	Parameter value as a BCD value.
624	AsInteger	rw	Get/Set parameter value as an integer (32-bit) value.
624	AsLargeInt	rw	Get/Set parameter value as a 64-bit integer value.
625	AsLongWord	rw	Get/Set parameter value as a 32-bit unsigned integer value.
625	AsMemo	rw	Get/Set parameter value as a memo (string) value.
625	AsShortInt	rw	
625	AsSingle	rw	
625	AsSmallInt	rw	Get/Set parameter value as a smallint value.
626	AsString	rw	Get/Set parameter value as a string value.
627	AsTime	rw	Get/Set parameter value as a time (TDateTime) value.
627	AsUnicodeString	rw	Parameter contents as a Unicode string.
626	AsUTF8String	rw	Parameter contents as an UTF8 string.
629	AsWideString	rw	Get/Set the value as a widestring.
627	AsWord	rw	Get/Set parameter value as a word value.
628	Bound	rw	Is the parameter value bound (set to fixed value).
628	Dataset	r	Dataset to which this parameter belongs.
629	DataType	rw	Data type of the parameter.
628	IsNull	r	Is the parameter empty.
629	Name	rw	Name of the parameter.
628	NativeStr	rw	No description available.
630	NumericScale	rw	Numeric scale.
630	ParamType	rw	Type of parameter.
630	Precision	rw	Precision of the BCD value.
631	Size	rw	Size of the parameter.
629	Text	rw	Read or write the value of the parameter as a string.
631	Value	rws	Value as a variant.

16.55.4 TParam.Create

Synopsis: Create a new parameter value.

Declaration:
```
constructor Create(ACollection: TCollection);  Override;  Overload
constructor Create(AParams: TParams; AParamType: TParamType);  Overload
                   ;  Reintroduce
```

Visibility: public

Description: `Create` first calls the inherited create, and then initializes the parameter properties. The first form creates a default parameter, the second form is a convenience function and initializes a parameter of a certain kind (`AParamType`), in which case the owning `TParams` collection must be specified in `AParams`

See also: TParams (631)

16.55.5 TParam.Assign

Synopsis: Assign one parameter instance to another.

Declaration: `procedure Assign(Source: TPersistent);  Override`

Visibility: public

Description: `Assign` copies the `Name`, `ParamType`, `Bound`, `Value`, `SizePrecision` and `NumericScale` properties from `ASource` if it is of type `TParam`. If `Source` is of type TField (549), then it is passed to TParam.AssignField (619). If `Source` is of type `TStrings`, then it is assigned to TParams.AsMemo (631).

Errors: If `Source` is not of type `TParam`, `TField` or `TStrings`, an exception will be raised.

See also: TField (549), TParam.Name (629), TParam.Bound (628), TParam.NumericScale (630), TParam.ParamType (630), TParam.value (631), TParam.Size (631), TParam.AssignField (619), Tparam.AsMemo (625)

16.55.6 TParam.AssignField

Synopsis: Copy value from field instance.

Declaration: `procedure AssignField(Field: TField)`

Visibility: public

Description: `AssignField` copies the `Field`, FieldName (569) and Value (566) to the parameter instance. The parameter is bound after this operation. If `Field` is `Nil` then the parameter name and value are cleared.

See also: TParam.assign (619), TParam.AssignToField (619), TParam.AssignFieldValue (619)

16.55.7 TParam.AssignToField

Synopsis: Assign parameter value to field.

Declaration: `procedure AssignToField(Field: TField)`

Visibility: public

Description: `AssignToField` copies the parameter value (631) to the `field` instance. If `Field` is `Nil`, nothing happens.

Errors: An EDatabaseError (458) exception is raised if the field has an unsupported field type (for types ftCursor, ftArray, ftDataset,ftReference).

See also: TParam.Assign (619), TParam.AssignField (619), TParam.AssignFromField (620)

16.55.8 TParam.AssignFieldValue

Synopsis: Assign field value to the parameter.

Declaration: `procedure AssignFieldValue(Field: TField; const AValue: Variant)`

Visibility: public

Description: `AssignFieldValue` copies only the field type from `Field` and the value from the `AValue` parameter. It sets the TParam.Bound (628) bound parameter to `True`. This method is called from TParam.AssignField (619).

See also: TField (549), TParam.AssignField (619), TParam.Bound (628)

16.55.9 TParam.AssignFromField

Synopsis: Copy field type and value.

Declaration: `procedure AssignFromField(Field: TField)`

Visibility: public

Description: `AssignFromField` copies the field value (566) and data type (TField.DataType (562)) to the parameter instance. If `Field` is `Nil`, nothing happens. This is the reverse operation of TParam.AssignToField (619).

Errors: An EDatabaseError (458) exception is raised if the field has an unsupported field type (for types ftCursor, ftArray, ftDataset,ftReference).

See also: TParam.Assign (619), TParam.AssignField (619), TParam.AssignToField (619)

16.55.10 TParam.Clear

Synopsis: Clear the parameter value.

Declaration: `procedure Clear`

Visibility: public

Description: `Clear` clears the parameter value, it is set to `UnAssigned`. The Datatype, parameter type or name are not touched.

See also: TParam.Value (631), TParam.Name (629), TParam.ParamType (630), TParam.DataType (629)

16.55.11 TParam.GetData

Synopsis: Get the parameter value from a memory buffer.

Declaration: `procedure GetData(Buffer: Pointer)`

Visibility: public

Description: `GetData` retrieves the parameter value and stores it in `buffer` It uses the same data layout as TField (549), and can be used to copy the parameter value to a record buffer.

Errors: Only basic field types are supported. Using an unsupported field type will result in an EdatabaseError (458) exception.

See also: TParam.SetData (621), TField (549)

16.55.12 TParam.GetDataSize

Synopsis: Return the size of the data.

Declaration: `function GetDataSize : Integer`

Visibility: public

Description: `GetDataSize` returns the size (in bytes) needed to store the current value of the parameter.

Errors: For an unsupported data type, an EDatabaseError (458) exception is raised when this function is called.

See also: TParam.GetData (620), TParam.SetData (621)

16.55.13 TParam.LoadFromFile

Synopsis: Load a parameter value from file.

Declaration: `procedure LoadFromFile(const FileName: string; BlobType: TBlobType)`

Visibility: public

Description: `LoadFromFile` can be used to load a BLOB-type parameter from a file named `FileName`. The `BlobType` parameter can be used to set the exact data type of the parameter: it must be one of the BLOB data types. This function simply creates a `TFileStream` instance and passes it to TParam.LoadFromStream (621).

Errors: If the specified `FileName` is not a valid file, or the file is not readable, an exception will occur.

See also: TParam.LoadFromStream (621), TBlobType (441), TParam.SaveToFile (617)

16.55.14 TParam.LoadFromStream

Synopsis: Load a parameter value from stream.

Declaration: `procedure LoadFromStream(Stream: TStream; BlobType: TBlobType)`

Visibility: public

Description: `LoadFromStream` can be used to load a BLOB-type parameter from a stream. The `BlobType` parameter can be used to set the exact data type of the parameter: it must be one of the BLOB data types.

Errors: If the stream does not support taking the `Size` of the stream, an exception will be raised.

See also: TParam.LoadFromFile (621), TParam.SaveToStream (617)

16.55.15 TParam.SetBlobData

Synopsis: Set BLOB data.

Declaration: `procedure SetBlobData(Buffer: Pointer; ASize: Integer)`

Visibility: public

Description: `SetBlobData` reads the value of a BLOB type parameter from a memory buffer: the data is read from the memory buffer `Buffer` and is assumed to be `Size` bytes long.

Errors: No checking is performed on the validity of the data buffer. If the data buffer is invalid or the size is wrong, an exception may occur.

See also: TParam.LoadFromStream (621)

16.55.16 TParam.SetData

Synopsis: Set the parameter value from a buffer.

Declaration: `procedure SetData(Buffer: Pointer)`

Visibility: public

Description: `SetData` performs the reverse operation of TParam.GetData (620): it reads the parameter value from the memory area pointed to by `Buffer`. The size of the data read is determined by TParam.GetDataSize (620) and the type of data by TParam.DataType (629) : it is the same storage mechanism used by TField (549), and so can be used to copy the value from a TDataset (496) record buffer.

Errors: Not all field types are supported. If an unsupported field type is encountered, an EDatabaseError (458) exception is raised.

See also: TDataset (496), TParam.GetData (620), TParam.DataType (629), TParam.GetDataSize (620)

16.55.17 TParam.AsBCD

Synopsis: Get or set parameter value as BCD value.

Declaration: `Property AsBCD : Currency`

Visibility: public

Access: Read,Write

Description: `AsBCD` can be used to get or set a parameter value as a BCD encoded floating point value.

See also: TParam.AsFloat (624)

16.55.18 TParam.AsBlob

Synopsis: Return parameter value as a blob.

Declaration: `Property AsBlob : TBlobData`

Visibility: public

Access: Read,Write

Description: `AsBlob` returns the parameter value as a blob: currently this is a string. It can be set to set the parameter value.

See also: TParam.AsString (626)

16.55.19 TParam.AsBoolean

Synopsis: Get/Set parameter value as a boolean value.

Declaration: `Property AsBoolean : Boolean`

Visibility: public

Access: Read,Write

Description: `AsBoolean` will return the parameter value as a boolean value. If it is written, the value is set to the specified value and the data type is set to `ftBoolean`.

See also: TParam.DataType (629), TParam.Value (631)

16.55.20 TParam.AsByte

Synopsis: Get/Set parameter value as a 8-bit unsigned integer value.

Declaration: `Property AsByte : LongInt`

Visibility: public

Access: Read,Write

Description: `AsByte` will return the parameter value as a 8-bit unsigned integer value. If it is written, the value is set to the specified value and the data type is set to `ftByte`.

See also: TParam.asInteger (624), TParam.asSmallint (625), TParam.AsWord (627), TParam.DataType (629), TParam.Value (631)

16.55.21 TParam.AsBytes

Synopsis: Get or set parameter value as `TBytes`.

Declaration: `Property AsBytes : TBytes`

Visibility: public

Access: Read,Write

Description: `AsBCD` can be used to get or set a parameter value as a `TBytes` value. This should normally only be used for blob type parameters.

See also: TParam.AsString (626)

16.55.22 TParam.AsCurrency

Synopsis: Get/Set parameter value as a currency value.

Declaration: `Property AsCurrency : Currency`

Visibility: public

Access: Read,Write

Description: `AsCurrency` will return the parameter value as a currency value. If it is written, the value is set to the specified value and the data type is set to `ftCurrency`.

See also: TParam.AsFloat (624), TParam.DataType (629), TParam.Value (631)

16.55.23 TParam.AsDate

Synopsis: Get/Set parameter value as a date (TDateTime) value.

Declaration: `Property AsDate : TDateTime`

Visibility: public

Access: Read,Write

Description: `AsDate` will return the parameter value as a date value. If it is written, the value is set to the specified value and the data type is set to `ftDate`.

See also: TParam.AsDateTime (624), TParam.AsTime (627), TParam.DataType (629), TParam.Value (631)

16.55.24 TParam.AsDateTime

Synopsis: Get/Set parameter value as a date/time (TDateTime) value.

Declaration: `Property AsDateTime : TDateTime`

Visibility: public

Access: Read,Write

Description: `AsDateTime` will return the parameter value as a TDateTime value. If it is written, the value is set to the specified value and the data type is set to `ftDateTime`.

See also: TParam.AsDate (623), TParam.asTime (627), TParam.DataType (629), TParam.Value (631)

16.55.25 TParam.AsFloat

Synopsis: Get/Set parameter value as a floating-point value.

Declaration: `Property AsFloat : Double`

Visibility: public

Access: Read,Write

Description: `AsFLoat` will return the parameter value as a double floating-point value. If it is written, the value is set to the specified value and the data type is set to `ftFloat`.

See also: TParam.AsCurrency (623), TParam.DataType (629), TParam.Value (631)

16.55.26 TParam.AsInteger

Synopsis: Get/Set parameter value as an integer (32-bit) value.

Declaration: `Property AsInteger : LongInt`

Visibility: public

Access: Read,Write

Description: `AsInteger` will return the parameter value as a 32-bit signed integer value. If it is written, the value is set to the specified value and the data type is set to `ftInteger`.

See also: TParam.AsLargeInt (624), TParam.AsSmallInt (625), TParam.AsWord (627), TParam.DataType (629), TParam.Value (631)

16.55.27 TParam.AsLargeInt

Synopsis: Get/Set parameter value as a 64-bit integer value.

Declaration: `Property AsLargeInt : LargeInt`

Visibility: public

Access: Read,Write

Description: `AsLargeInt` will return the parameter value as a 64-bit signed integer value. If it is written, the value is set to the specified value and the data type is set to `ftLargeInt`.

See also: TParam.asInteger (624), TParam.asSmallint (625), TParam.AsWord (627), TParam.DataType (629), TParam.Value (631)

16.55.28 TParam.AsLongWord

Synopsis: Get/Set parameter value as a 32-bit unsigned integer value.

Declaration: `Property AsLongWord : LongWord`

Visibility: public

Access: Read,Write

Description: `AsLongWorde` will return the parameter value as a 32-bit unsigned integer value. If it is written, the value is set to the specified value and the data type is set to `ftLongWord`.

See also: TParam.asInteger (624), TParam.asSmallint (625), TParam.AsWord (627), TParam.DataType (629), TParam.Value (631)

16.55.29 TParam.AsMemo

Synopsis: Get/Set parameter value as a memo (string) value.

Declaration: `Property AsMemo : string`

Visibility: public

Access: Read,Write

Description: `AsMemo` will return the parameter value as a memo (string) value. If it is written, the value is set to the specified value and the data type is set to `ftMemo`.

See also: TParam.asString (626), TParam.LoadFromStream (621), TParam.SaveToStream (617), TParam.DataType (629), TParam.Value (631)

16.55.30 TParam.AsShortInt

Declaration: `Property AsShortInt : LongInt`

Visibility: public

Access: Read,Write

16.55.31 TParam.AsSingle

Declaration: `Property AsSingle : Single`

Visibility: public

Access: Read,Write

16.55.32 TParam.AsSmallInt

Synopsis: Get/Set parameter value as a smallint value.

Declaration: `Property AsSmallInt : LongInt`

Visibility: public

Access: Read,Write

Description: `AsSmallint` will return the parameter value as a 16-bit signed integer value. If it is written, the value is set to the specified value and the data type is set to `ftSmallint`.

See also: TParam.AsInteger (624), TParam.AsLargeInt (624), TParam.AsWord (627), TParam.DataType (629), TParam.Value (631)

16.55.33 TParam.AsString

Synopsis: Get/Set parameter value as a string value.

Declaration: `Property AsString : string`

Visibility: public

Access: Read,Write

Description: `AsString` will return the parameter value as a string value. If it is written, the value is set to the specified value and the data type is set to `ftString`.

See also: TParam.DataType (629), TParam.Value (631)

16.55.34 TParam.AsAnsiString

Synopsis: Parameter contents as an ANSI string.

Declaration: `Property AsAnsiString : AnsiString`

Visibility: public

Access: Read,Write

Description: `AsAnsiString` returns the parameter data as an ANSI string (single byte character string). Note that if the parameter contains unicode data, some characters may get lost when reading.

See also: TParam.AsString (626), TParam.AsUnicodeString (627), TParam.AsUTF8String (626)

16.55.35 TParam.AsUTF8String

Synopsis: Parameter contents as an UTF8 string.

Declaration: `Property AsUTF8String : UTF8String`

Visibility: public

Access: Read,Write

Description: `AsUTF8String` returns the parameter data as an UTF8 string (single byte-encoded unicode string).

See also: TParam.AsString (626), TParam.AsUnicodeString (627), TParam.AsAnsiString (626)

16.55.36 TParam.AsUnicodeString

Synopsis: Parameter contents as a Unicode string.

Declaration: `Property AsUnicodeString : UnicodeString`

Visibility: public

Access: Read,Write

Description: `AsUTF8String` returns the parameter data as a UnicodeString (double byte unicode string).

See also: TParam.AsString (626), TParam.AsUTF8String (626), TParam.AsAnsiString (626)

16.55.37 TParam.AsTime

Synopsis: Get/Set parameter value as a time (TDateTime) value.

Declaration: `Property AsTime : TDateTime`

Visibility: public

Access: Read,Write

Description: `AsTime` will return the parameter value as a time (TDateTime) value. If it is written, the value is set to the specified value and the data type is set to `ftTime`.

See also: TParam.AsDate (623), TParam.AsDateTime (624), TParam.DataType (629), TParam.Value (631)

16.55.38 TParam.AsWord

Synopsis: Get/Set parameter value as a word value.

Declaration: `Property AsWord : LongInt`

Visibility: public

Access: Read,Write

Description: `AsWord` will return the parameter value as an integer. If it is written, the value is set to the specified value and the data type is set to `ftWord`.

See also: TParam.AsInteger (624), TParam.AsLargeInt (624), TParam.AsSmallint (625), TParam.DataType (629), TParam.Value (631)

16.55.39 TParam.AsFMTBCD

Synopsis: Parameter value as a BCD value.

Declaration: `Property AsFMTBCD : TBCD`

Visibility: public

Access: Read,Write

Description: `AsFMTBCD` can be used to get or set the parameter's value as a BCD typed value.

See also: AsFloat (438), AsCurrency (438)

16.55.40 TParam.Bound

Synopsis: Is the parameter value bound (set to fixed value).

Declaration: `Property Bound : Boolean`

Visibility: public

Access: Read,Write

Description: `Bound` indicates whether a parameter has received a fixed value: setting the parameter value will set `Bound` to `True`. When creating master-detail relationships, parameters with their `Bound` property set to `True` will not receive a value from the master dataset: their value will be kept. Only parameters where `Bound` is `False` will receive a new value from the master dataset.

See also: TParam.DataType (629), TParam.Value (631)

16.55.41 TParam.Dataset

Synopsis: Dataset to which this parameter belongs.

Declaration: `Property Dataset : TDataSet`

Visibility: public

Access: Read

Description: `Dataset` is the dataset that owns the TParams (631) instance of which this `TParam` instance is a part. It is `Nil` if the collection is not set, or is not a `TParams` instance.

See also: TDataset (496), TParams (631)

16.55.42 TParam.IsNull

Synopsis: Is the parameter empty.

Declaration: `Property IsNull : Boolean`

Visibility: public

Access: Read

Description: `IsNull` is `True` is the value is empty or not set (Null or UnAssigned).

See also: TParam.Clear (620), TParam.Value (631)

16.55.43 TParam.NativeStr

Synopsis: No description available.

Declaration: `Property NativeStr : string`

Visibility: public

Access: Read,Write

Description: No description available

16.55.44 TParam.Text

Synopsis: Read or write the value of the parameter as a string.

Declaration: `Property Text : string`

Visibility: public

Access: Read,Write

Description: `AsText` returns the same value as TParam.AsString (626), but, when written, does not set the data type: instead, it attempts to convert the value to the type specified in TParam.Datatype (629).

See also: TParam.AsString (626), TParam.DataType (629)

16.55.45 TParam.AsWideString

Synopsis: Get/Set the value as a widestring.

Declaration: `Property AsWideString : WideString`

Visibility: public

Access: Read,Write

Description: `AsWideString` returns the parameter value as a widestring value. Setting the property will set the value of the parameter and will also set the DataType (629) to `ftWideString`.

See also: TParam.AsString (626), TParam.Value (631), TParam.DataType (629)

16.55.46 TParam.DataType

Synopsis: Data type of the parameter.

Declaration: `Property DataType : TFieldType`

Visibility: published

Access: Read,Write

Description: `DataType` is the current data type of the parameter value. It is set automatically when one of the various `AsXYZ` properties is written, or when the value is copied from a field value.

See also: TParam.IsNull (628), TParam.Value (631), TParam.AssignField (619)

16.55.47 TParam.Name

Synopsis: Name of the parameter.

Declaration: `Property Name : string`

Visibility: published

Access: Read,Write

Description: `Name` is the name of the parameter. The name is usually determined automatically from the SQL statement the parameter is part of. Each parameter name should appear only once in the collection.

See also: TParam.DataType (629), TParam.Value (631), TParams.ParamByName (634)

16.55.48 TParam.NumericScale

Synopsis: Numeric scale.

Declaration: `Property NumericScale : Integer`

Visibility: published

Access: Read,Write

Description: `NumericScale` can be used to store the numerical scale for BCD values. It is currently unused.

See also: TParam.Precision (630), TParam.Size (631)

16.55.49 TParam.ParamType

Synopsis: Type of parameter.

Declaration: `Property ParamType : TParamType`

Visibility: published

Access: Read,Write

Description: `ParamTyp` specifies the type of parameter: is the parameter value written to the database engine, or is it received from the database engine, or both ? It can have the following value:

ptUnknownUnknown type.

ptInputInput parameter.

ptOutputOutput parameter, filled on result.

ptInputOutputInput/output parameter.

ptResultResult parameter.

The `ParamType` property is usually set by the database engine that creates the parameter instances.

See also: TParam.DataType (629), TParam.DataSize (617), TParam.Name (629)

16.55.50 TParam.Precision

Synopsis: Precision of the BCD value.

Declaration: `Property Precision : Integer`

Visibility: published

Access: Read,Write

Description: `Precision` can be used to store the numerical precision for BCD values. It is currently unused.

See also: TParam.NumericScale (630), TParam.Size (631)

16.55.51 TParam.Size

Synopsis: Size of the parameter.

Declaration: `Property Size : Integer`

Visibility: published

Access: Read,Write

Description: `Size` is the declared size of the parameter. In the current implementation, this parameter is ignored other than copying it from TField.DataSize (562) in the TParam.AssignFieldValue (619) method. The actual size can be retrieved through the TParam.Datasize (617) property.

See also: TParam.Datasize (617), TField.DataSize (562), TParam.AssignFieldValue (619)

16.55.52 TParam.Value

Synopsis: Value as a variant.

Declaration: `Property Value : Variant`

Visibility: published

Access: Read,Write

Description: `Value` returns (or sets) the value as a variant value.

See also: TParam.DataType (629)

16.56 TParams

16.56.1 Description

`TParams` is a collection of TParam (617) values. It is used to specify parameter values for parameterized SQL statements, but is also used to specify parameter values for stored procedures. Its default property is an array of TParam (617) values. The class also offers a method to scan a SQL statement for parameter names and replace them with placeholders understood by the SQL engine: TParams.ParseSQL (634).

TDataset (496) itself does not use `TParams`. The class is provided in the DB unit, so all `TDataset` descendents that need some kind of parameterization make use of the same interface. The TMasterParamsDataLink (611) class can be used to establish a master-detail relationship between a parameter-aware `TDataset` instance and another dataset; it will automatically refresh parameter values when the fields in the master dataset change. To this end, the TParams.CopyParamValuesFromDataset (636) method exists.

See also: TDataset (496), TMasterParamsDataLink (611), TParam (617), TParams.ParseSQL (634), TParams.CopyParamValuesFromDa (636)

16.56.2 Method overview

Page	Method	Description
632	AddParam	Add a parameter to the collection.
633	AssignValues	Copy values from another collection.
636	CopyParamValuesFromDataset	Copy parameter values from the fields in a dataset.
632	Create	Create a new instance of `TParams`.
633	CreateParam	Create and add a new parameter to the collection.
633	FindParam	Find a parameter with given name.
634	GetEnumerator	Return an enumerator for the parameters.
633	GetParamList	Fetch a list of `TParam` instances.
634	IsEqual	Is the list of parameters equal.
634	ParamByName	Return a parameter by name.
634	ParseSQL	Parse SQL statement, replacing parameter names with SQL parameter placeholders.
635	RemoveParam	Remove a parameter from the collection.

16.56.3 Property overview

Page	Properties	Access	Description
636	Dataset	r	Dataset that owns the `TParams` instance.
636	Items	rw	Indexed access to `TParams` instances in the collection.
636	ParamValues	rw	Named access to the parameter values.

16.56.4 TParams.Create

Synopsis: Create a new instance of `TParams`.

Declaration:
```
constructor Create(AOwner: TPersistent;
                   AItemClass: TCollectionItemClass);  Overload
constructor Create(AOwner: TPersistent);  Overload
constructor Create;  Overload
```

Visibility: public

Description: `Create` initializes a new instance of `TParams`. It calls the inherited constructor with TParam (617) as the collection's item class, and sets `AOwner` as the owner of the collection. Usually, `AOwner` will be the dataset that needs parameters.

See also: #rtl.classes.TCollection.create (**??**), TParam (617)

16.56.5 TParams.AddParam

Synopsis: Add a parameter to the collection.

Declaration:
```
procedure AddParam(Value: TParam)
```

Visibility: public

Description: `AddParam` adds `Value` to the collection.

Errors: No checks are done on the `TParam` instance. If it is `Nil`, an exception will be raised.

See also: TParam (617), #rtl.classes.tcollection.add (**??**)

16.56.6 TParams.AssignValues

Synopsis: Copy values from another collection.

Declaration: `procedure AssignValues(Value: TParams)`

Visibility: public

Description: `AssignValues` examines all TParam (617) instances in `Value`, and looks in its own items for a `TParam` instance with the same name. If it is found, then the value and type of the parameter are copied (using TParam.Assign (619)). If it is not found, nothing is done.

See also: TParam (617), TParam.Assign (619)

16.56.7 TParams.CreateParam

Synopsis: Create and add a new parameter to the collection.

Declaration:
```
function CreateParam(FldType: TFieldType; const ParamName: string;
                     ParamType: TParamType) : TParam
```

Visibility: public

Description: `CreateParam` creates a new TParam (617) instance with datatype equal to `fldType`, `Name` equal to `ParamName` and sets its `ParamType` property to `ParamType`. The parameter is then added to the collection.

See also: TParam (617), TParam.Name (629), TParam.Datatype (629), TParam.Paramtype (630)

16.56.8 TParams.FindParam

Synopsis: Find a parameter with given name.

Declaration: `function FindParam(const Value: string) : TParam`

Visibility: public

Description: `FindParam` searches the collection for the TParam (617) instance with property `Name` equal to `Value`. It will return the last instance with the given name, and will only return one instance. If no match is found, `Nil` is returned.

Remark A `TParams` collection can have 2 `TParam` instances with the same name: no checking for duplicates is done.

See also: TParam.Name (629), TParams.ParamByName (634), TParams.GetParamList (633)

16.56.9 TParams.GetParamList

Synopsis: Fetch a list of `TParam` instances.

Declaration: `procedure GetParamList(List: TList; const ParamNames: string)`

Visibility: public

Description: `GetParamList` examines the parameter names in the semicolon-separated list `ParamNames`. It searches each `TParam` instance from the names in the list and adds it to `List`.

Errors: If the `ParamNames` list contains an unknown parameter name, then an exception is raised. Whitespace is not discarded.

See also: TParam (617), TParam.Name (629), TParams.ParamByName (634)

16.56.10 TParams.IsEqual

Synopsis: Is the list of parameters equal.

Declaration:
```
function IsEqual(Value: TParams) : Boolean
```

Visibility: public

Description: `IsEqual` compares the parameter count of `Value` and if it matches, it compares all `TParam` items of `Value` with the items it owns. If all items are equal (all properties match), then `True` is returned. The items are compared on index, so the order is important.

See also: TParam (617)

16.56.11 TParams.GetEnumerator

Synopsis: Return an enumerator for the parameters.

Declaration:
```
function GetEnumerator : TParamsEnumerator
```

Visibility: public

Description: `GetEnumerator` returns an enumerator that loops over all parameters (as implemented by TParamsEnumerator (637))

See also: TParamsEnumerator (637)

16.56.12 TParams.ParamByName

Synopsis: Return a parameter by name.

Declaration:
```
function ParamByName(const Value: string) : TParam
```

Visibility: public

Description: `ParamByName` searches the collection for the TParam (617) instance with property `Name` equal to `Value`. It will return the last instance with the given name, and will only return one instance. If no match is found, an exception is raised.

Remark A `TParams` collection can have 2 `TParam` instances with the same name: no checking for duplicates is done.

See also: TParam.Name (629), TParams.FindParam (633), TParams.GetParamList (633)

16.56.13 TParams.ParseSQL

Synopsis: Parse SQL statement, replacing parameter names with SQL parameter placeholders.

Declaration:
```
function ParseSQL(const SQL: string; DoCreate: Boolean) : string
                  ;  Overload
function ParseSQL(const SQL: string; DoCreate: Boolean;
                  EscapeSlash: Boolean; EscapeRepeat: Boolean;
                  ParameterStyle: TParamStyle) : string;  Overload
function ParseSQL(const SQL: string; DoCreate: Boolean;
                  EscapeSlash: Boolean; EscapeRepeat: Boolean;
                  ParameterStyle: TParamStyle;
                  out ParamBinding: TParamBinding) : string;  Overload
function ParseSQL(const SQL: string; DoCreate: Boolean;
```

```
                    EscapeSlash: Boolean; EscapeRepeat: Boolean;
                    ParameterStyle: TParamStyle;
                    out ParamBinding: TParamBinding;
                    out ReplaceString: string) : string;  Overload
function ParseSQL(const SQL: string; Options: TSQLParseOptions;
                    ParameterStyle: TParamStyle;
                    out ParamBinding: TParamBinding; MacroChar: Char;
                    out ReplaceString: string) : string
```

Visibility: public

Description: `ParseSQl` parses the `SQL` statement for parameter names in the form `:ParamName`. It replaces them with a SQL parameter placeholder. If `DoCreate` is `True` then a `TParam` instance is added to the collection with the found parameter name.

The parameter placeholder is determined by the `ParameterStyle` property, which can have the following values:

psInterbaseParameters are specified by a ? character.

psPostgreSQLParameters are specified by a $N character.

psSimulatedParameters are specified by a $N character.

`psInterbase` is the default.

If the `EscapeSlash` parameter is `True`, then backslash characters are used to quote the next character in the SQL statement. If it is `False`, the backslash character is regarded as a normal character.

If the `EscapeRepeat` parameter is `True` (the default) then embedded quotes in string literals are escaped by repeating themselves. If it is `false` then they should be quoted with backslashes.

`ParamBinding`, if specified, is filled with the indexes of the parameter instances in the parameter collection: for each SQL parameter placeholder, the index of the corresponding `TParam` instance is returned in the array.

`ReplaceString`, if specified, contains the placeholder used for the parameter names (by default, $). It has effect only when `ParameterStyle` equals `psSimulated`.

The function returns the SQL statement with the parameter names replaced by placeholders.

See also: TParam (617), TParam.Name (629), TParamStyle (451)

16.56.14 TParams.RemoveParam

Synopsis: Remove a parameter from the collection.

Declaration: `procedure RemoveParam(Value: TParam)`

Visibility: public

Description: `RemoveParam` removes the parameter `Value` from the collection, but does not free the instance.

Errors: `Value` must be a valid instance, or an exception will be raised.

See also: TParam (617)

16.56.15 TParams.CopyParamValuesFromDataset

Synopsis: Copy parameter values from the fields in a dataset.

Declaration:
```
procedure CopyParamValuesFromDataset(ADataset: TDataSet;
                                     CopyBound: Boolean)
```

Visibility: public

Description: `CopyParamValuesFromDataset` assigns values to all parameters in the collection by searching in `ADataset` for fields with the same name, and assigning the value of the field to the `Tparam` instances using TParam.AssignField (619). By default, this operation is only performed on `TParam` instances with their Bound (628) property set to `False`. If `CopyBound` is true, then the operation is performed on all `TParam` instances in the collection.

Errors: If, for some `TParam` instance, `ADataset` misses a field with the same name, an `EDatabaseError` exception will be raised.

See also: TParam (617), TParam.Bound (628), TParam.AssignField (619), TDataset (496), TDataset.FieldByName (508)

16.56.16 TParams.Dataset

Synopsis: Dataset that owns the `TParams` instance.

Declaration:
```
Property Dataset : TDataSet
```

Visibility: public

Access: Read

Description: `Dataset` is the TDataset (496) instance that was specified when the `TParams` instance was created.

See also: TParams.Create (632), TDataset (496)

16.56.17 TParams.Items

Synopsis: Indexed access to `TParams` instances in the collection.

Declaration:
```
Property Items[Index: Integer]: TParam; default
```

Visibility: public

Access: Read,Write

Description: `Items` is overridden by `TParams` so it has the proper type (`TParam`). The `Index` runs from 0 to `Count-1`.

See also: TParams (631)

16.56.18 TParams.ParamValues

Synopsis: Named access to the parameter values.

Declaration:
```
Property ParamValues[ParamName: string]: Variant
```

Visibility: public

Access: Read,Write

Description: `ParamValues` provides access to the parameter values (TParam.Value (631)) by name. It is equivalent to reading and writing

```
ParamByName(ParamName).Value
```

See also: TParam.Value (631), TParams.ParamByName (634)

16.57 TParamsEnumerator

16.57.1 Description

`TParamsEnumerator` is a helper class to implement enumeration (`for..in`) of parameters. It implements the `IEnumerator` interface.

See also: TParams.GetEnumerator (634)

16.57.2 Method overview

Page	Method	Description
637	Create	Create a new `TParamsEnumerator` instance.
637	MoveNext	Go to next TParam.

16.57.3 Property overview

Page	Properties	Access	Description
638	Current	r	Current `TParam` instance.

16.57.4 TParamsEnumerator.Create

Synopsis: Create a new `TParamsEnumerator` instance.

Declaration: `constructor Create(AParams: TParams)`

Visibility: public

Description: `Create` instantiates a new enumerator for `AParams`.

See also: TParams.GetEnumerator (634)

16.57.5 TParamsEnumerator.MoveNext

Synopsis: Go to next TParam.

Declaration: `function MoveNext : Boolean`

Visibility: public

Description: `MoveNext` will move to the next TParam instance if possible. If it returns `True` then TParamsEnumerator.Current (638) will return the new current `TParam`

See also: TParamsEnumerator.Current (638)

16.57.6 TParamsEnumerator.Current

Synopsis: Current `TParam` instance.

Declaration: `Property Current : TParam`

Visibility: public

Access: Read

Description: `Current` is the current `TParam` instance. It is only valid if TParamsEnumerator.MoveNext (637) returned true.

See also: TParamsEnumerator.MoveNext (637)

16.58 TShortintField

16.58.1 Description

`TShortIntField` is instantiated when a dataset must manage a field with 8-bit signed data: the data type `ftShortInt`. It overrides some methods of TField (549) to handle `ShortInt` data, and sets some of the properties to values for `ShortInt` data. It also introduces some methods and properties specific to integer data such as MinValue (605) and MaxValue (605).

It should never be necessary to create an instance of `TShortIntField` manually, a field of this class will be instantiated automatically for each integer field when a dataset is opened.

See also: MinValue (605), MaxValue (605)

16.58.2 Method overview

Page	Method	Description
638	Create	Create new instance of `TShortintField`.

16.58.3 TShortintField.Create

Synopsis: Create new instance of `TShortintField`.

Declaration: `constructor Create(AOwner: TComponent); Override`

Visibility: public

Description: `Create` calls the inherited constructor and sets the values of the MinValue (605)MaxValue (605) and TField.DataType (562) properties.

See also: MinValue (605), MaxValue (605), TField.DataType (562)

16.59 TSingleField

16.59.1 Method overview

Page	Method	Description
639	CheckRange	
639	Create	

16.59.2 Property overview

Page	Properties	Access	Description
639	Currency	rw	
639	MaxValue	rw	
639	MinValue	rw	
640	Precision	rw	
639	Value	rw	

16.59.3 TSingleField.Create

Declaration: `constructor Create(AOwner: TComponent);  Override`

Visibility: public

16.59.4 TSingleField.CheckRange

Declaration: `function CheckRange(AValue: Single) : Boolean`

Visibility: public

16.59.5 TSingleField.Value

Declaration: `Property Value : Single`

Visibility: public

Access: Read,Write

16.59.6 TSingleField.Currency

Declaration: `Property Currency : Boolean`

Visibility: published

Access: Read,Write

16.59.7 TSingleField.MaxValue

Declaration: `Property MaxValue : Single`

Visibility: published

Access: Read,Write

16.59.8 TSingleField.MinValue

Declaration: `Property MinValue : Single`

Visibility: published

Access: Read,Write

16.59.9 TSingleField.Precision

Declaration: `Property Precision : LongInt`

Visibility: published

Access: Read,Write

16.60 TSmallintField

16.60.1 Description

`TSmallIntField` is the class created when a dataset must manage 16-bit signed integer data, of datatype `ftSmallInt`. It exposes no new properties, but simply overrides some methods to manage 16-bit signed integer data.

It should never be necessary to create an instance of `TSmallIntField` manually, a field of this class will be instantiated automatically for each smallint field when a dataset is opened.

See also: TField (549), TNumericField (614), TLongintField (603), TWordField (647)

16.60.2 Method overview

Page	Method	Description
640	Create	Create a new instance of the `TSmallintField` class.

16.60.3 TSmallintField.Create

Synopsis: Create a new instance of the `TSmallintField` class.

Declaration: `constructor Create(AOwner: TComponent); Override`

Visibility: public

Description: `Create` initializes a new instance of the TSmallintField (640) class. It calls the inherited constructor and then simply sets some of the TField (549) properties to work with 16-bit signed integer data.

See also: TField (549)

16.61 TStringField

16.61.1 Description

`TStringField` is the class used whenever a dataset has to handle a string field type (data type `ftString`). This class overrides some of the standard TField (549) methods to handle string data, and introduces some properties that are only pertinent for data fields of string type. It should never be necessary to create an instance of `TStringField` manually, a field of this class will be instantiated automatically for each string field when a dataset is opened.

See also: TField (549), TWideStringField (646), TDataset (496)

16.61.2 Method overview

Page	Method	Description
641	Create	Create a new instance of the TStringField class.
641	SetFieldType	Set the field type.

16.61.3 Property overview

Page	Properties	Access	Description
641	CodePage	r	Codepage of the field string data.
642	EditMask		Specify an edit mask for an edit control.
642	FixedChar	rw	Is the string declared with a fixed length ?
643	Size		Maximum size of the string.
642	Transliterate	rw	Should the field value be transliterated when reading or writing.
642	Value	rw	Value of the field as a string.

16.61.4 TStringField.Create

Synopsis: Create a new instance of the TStringField class.

Declaration: `constructor Create(AOwner: TComponent);  Override`

Visibility: public

Description: `Create` is used to create a new instance of the `TStringField` class. It initializes some TField (549) properties after having called the inherited constructor.

16.61.5 TStringField.SetFieldType

Synopsis: Set the field type.

Declaration: `procedure SetFieldType(AValue: TFieldType);  Override`

Visibility: public

Description: `SetFieldType` is overridden in TStringField (640) to check the data type more accurately (`ftString` and `ftFixedChar`). No extra functionality is added.

See also: TField.DataType (562)

16.61.6 TStringField.CodePage

Synopsis: Codepage of the field string data.

Declaration: `Property CodePage : TSystemCodePage`

Visibility: public

Access: Read

Description: `CodePage` is the code page of the string data in the field. It is determined when the field is initially created from the dataset's data, and cannot be changed while the dataset is active.

See also: TField.AsString (559), TField.AsUnicodeString (560), TField.AsAnsi8String (549), TFieldDef.CodePage (579)

16.61.7 TStringField.FixedChar

Synopsis: Is the string declared with a fixed length ?

Declaration: `Property FixedChar : Boolean`

Visibility: public

Access: Read,Write

Description: `FixedChar` is `True` if the underlying data engine has declared the field with a fixed length, as in a SQL CHAR() declaration: the field's value will then always be padded with as many spaces as needed to obtain the declared length of the field. If it is `False` then the declared length is simply the maximum length for the field, and no padding with spaces is performed.

16.61.8 TStringField.Transliterate

Synopsis: Should the field value be transliterated when reading or writing.

Declaration: `Property Transliterate : Boolean`

Visibility: public

Access: Read,Write

Description: `Transliterate` can be set to `True` if the field's contents should be transliterated prior to copying it from or to the field's buffer. Transliteration is done by a method of `TDataset`: TDataset.Translate (517).

See also: TDataset.Translate (517)

16.61.9 TStringField.Value

Synopsis: Value of the field as a string.

Declaration: `Property Value : string`

Visibility: public

Access: Read,Write

Description: `Value` is overridden in `TField` to return the value of the field as a string. It returns the contents of TField.AsString (559) when read, or sets the `AsString` property when written to.

See also: TField.AsString (559), TField.Value (566)

16.61.10 TStringField.EditMask

Synopsis: Specify an edit mask for an edit control.

Declaration: `Property EditMask :`

Visibility: published

Access:

Description: `EditMask` can be used to specify an edit mask for controls that allow to edit this field. It has no effect on the field value, and serves only to ensure that the user can enter only correct data for this field.

`TStringField` just changes the visibility of the `EditMark` property, it is introduced in `TField`.

For more information on valid edit masks, see the documentation of the GUI controls.

See also: TField.EditMask (563)

16.61.11 TStringField.Size

Synopsis: Maximum size of the string.

Declaration: `Property Size :`

Visibility: published

Access:

Description: `Size` is made published by the `TStringField` class so it can be set in the IDE: it is the declared maximum size of the string (in characters) and is used to calculate the size of the dataset buffer.

See also: TField.Size (565)

16.62 TTimeField

16.62.1 Description

`TimeField` is the class used when a dataset must manage data of type time. (TField.DataType (562) equals `ftTime`). It initializes some of the properties of the TField (549) class to be able to work with time fields.

It should never be necessary to create an instance of `TTimeField` manually, a field of this class will be instantiated automatically for each time field when a dataset is opened.

See also: TDataset (496), TField (549), TDateTimeField (540), TDateField (540)

16.62.2 Method overview

Page	Method	Description
643	Create	Create a new instance of a `TTimeField` class.

16.62.3 TTimeField.Create

Synopsis: Create a new instance of a `TTimeField` class.

Declaration: `constructor Create(AOwner: TComponent); Override`

Visibility: public

Description: `Create` initializes a new instance of the `TTimeField` class. It calls the inherited destructor, and then sets some TField (549) properties to configure the instance for working with time values.

See also: TField (549)

16.63 TVarBytesField

16.63.1 Description

`TVarBytesField` is the class used when a dataset must manage data of variable-size binary type. (TField.DataType (562) equals `ftVarBytes`). It initializes some of the properties of the TField (549) class to be able to work with variable-size byte fields.

It should never be necessary to create an instance of `TVarBytesField` manually, a field of this class will be instantiated automatically for each variable-sized binary data field when a dataset is opened.

See also: TDataset (496), TField (549), TBytesField (478)

16.63.2 Method overview

Page	Method	Description
644	Create	Create a new instance of a `TVarBytesField` class.

16.63.3 TVarBytesField.Create

Synopsis: Create a new instance of a `TVarBytesField` class.

Declaration: `constructor Create(AOwner: TComponent); Override`

Visibility: public

Description: `Create` initializes a new instance of the `TVarBytesField` class. It calls the inherited destructor, and then sets some TField (549) properties to configure the instance for working with variable-size binary data values.

See also: TField (549)

16.64 TVariantField

16.64.1 Description

`TVariantField` is the class used when a dataset must manage native variant-typed data. (TField.DataType (562) equals `ftVariant`). It initializes some of the properties of the TField (549) class and overrides some of its methods to be able to work with variant data.

It should never be necessary to create an instance of `TVariantField` manually, a field of this class will be instantiated automatically for each variant field when a dataset is opened.

See also: TDataset (496), TField (549)

16.64.2 Method overview

Page	Method	Description
645	Create	Create a new instance of the `TVariantField` class.

16.64.3 TVariantField.Create

Synopsis: Create a new instance of the `TVariantField` class.

Declaration: `constructor Create(AOwner: TComponent); Override`

Visibility: public

Description: `Create` initializes a new instance of the `TVariantField` class. It calls the inherited destructor, and then sets some TField (549) properties to configure the instance for working with variant values.

See also: TField (549)

16.65 TWideMemoField

16.65.1 Description

`TWideMemoField` is the class used when a dataset must manage memo (Text BLOB) data. (TField.DataType (562) equals `ftWideMemo`). It initializes some of the properties of the TField (549) class. All methods to be able to work with widestring memo fields have been implemented in the TBlobField (471) parent class.

It should never be necessary to create an instance of `TWideMemoField` manually, a field of this class will be instantiated automatically for each widestring memo field when a dataset is opened.

See also: TDataset (496), TField (549), TBLOBField (471), TMemoField (612), TGraphicField (595)

16.65.2 Method overview

Page	Method	Description
645	Create	Create a new instance of the `TWideMemoField` class.

16.65.3 Property overview

Page	Properties	Access	Description
646	Value	rw	Value of the field's contents as a widestring.

16.65.4 TWideMemoField.Create

Synopsis: Create a new instance of the `TWideMemoField` class.

Declaration: `constructor Create(aOwner: TComponent); Override`

Visibility: public

Description: `Create` initializes a new instance of the `TWideMemoField` class. It calls the inherited destructor, and then sets some TField (549) properties to configure the instance for working with widestring memo values.

See also: TField (549)

16.65.5 TWideMemoField.Value

Synopsis: Value of the field's contents as a widestring.

Declaration: `Property Value : WideString`

Visibility: public

Access: Read,Write

Description: `Value` is redefined by `TWideMemoField` as a WideString value. Reading and writing this property is equivalent to reading and writing the TField.AsWideString (560) property.

See also: TField.Value (566), Tfield.AsWideString (560)

16.66 TWideStringField

16.66.1 Description

`TWideStringField` is the string field class instantiated for fields of data type `ftWideString`. This class overrides some of the standard TField (549) methods to handle widestring data, and introduces some properties that are only pertinent for data fields of widestring type. It should never be necessary to create an instance of `TWideStringField` manually, a field of this class will be instantiated automatically for each widestring field when a dataset is opened.

See also: TField (549), TStringField (640), TDataset (496)

16.66.2 Method overview

Page	Method	Description
646	Create	Create a new instance of the `TWideStringField` class.
646	SetFieldType	Set the field type.

16.66.3 Property overview

Page	Properties	Access	Description
647	Value	rw	Value of the field as a widestring.

16.66.4 TWideStringField.Create

Synopsis: Create a new instance of the `TWideStringField` class.

Declaration: `constructor Create(AOwner: TComponent); Override`

Visibility: public

Description: `Create` is used to create a new instance of the `TWideStringField` class. It initializes some TField (549) properties after having called the inherited constructor.

16.66.5 TWideStringField.SetFieldType

Synopsis: Set the field type.

Declaration: `procedure SetFieldType(AValue: TFieldType); Override`

Visibility: public

Description: `SetFieldType` is overridden in TWideStringField (646) to check the data type more accurately (`ftWideString` and `ftFixedWideChar`). No extra functionality is added.

See also: TField.DataType (562)

16.66.6 TWideStringField.Value

Synopsis: Value of the field as a widestring.

Declaration: `Property Value : WideString`

Visibility: public

Access: Read,Write

Description: `Value` is overridden by the `TWideStringField` to return a `WideString` value. It is the same value as the TField.AsWideString (560) property.

See also: TField.AsWideString (560), TField.Value (566)

16.67 TWordField

16.67.1 Description

`TWordField` is the class created when a dataset must manage 16-bit unsigned integer data, of datatype `ftWord`. It exposes no new properties, but simply overrides some methods to manage 16-bit unsigned integer data.

It should never be necessary to create an instance of `TWordField` manually, a field of this class will be instantiated automatically for each word field when a dataset is opened.

See also: TField (549), TNumericField (614), TLongintField (603), TSmallIntField (640)

16.67.2 Method overview

Page	Method	Description
647	Create	Create a new instance of the `TWordField` class.

16.67.3 TWordField.Create

Synopsis: Create a new instance of the `TWordField` class.

Declaration: `constructor Create(AOwner: TComponent); Override`

Visibility: public

Description: `Create` initializes a new instance of the TWordField (647) class. It calls the inherited constructor and then simply sets some of the TField (549) properties to work with 16-bit unsigned integer data.

See also: TField (549)

Chapter 17

Reference for unit 'dbugintf'

17.1 Used units

Table 17.1: Used units by unit 'dbugintf'

Name	Page
dbugmsg	655
System	??

17.2 Overview

Use dbugintf to add debug messages to your application. The messages are not sent to standard output, but are sent to a debug server process which collects messages from various clients and displays them somehow on screen.

The unit is transparent in its use: it does not need initialization, it will start the debug server by itself if it can find it: the program should be called debugserver and should be in the `PATH`. When the first debug message is sent, the unit will initialize itself.

The FCL contains a sample debug server (dbugsvr) which can be started in advance, and which writes debug message to the console (both on Windows and Linux). The Lazarus project contains a visual application which displays the messages in a GUI.

The dbugintf unit relies on the SimpleIPC (648) mechanism to communicate with the debug server, hence it works on all platforms that have a functional version of that unit. It also uses `TProcess` to start the debug server if needed, so the process (648) unit should also be functional.

17.3 Writing a debug server.

Writing a debug server is relatively easy. It should instantiate a `TSimpleIPCServer` class from the SimpleIPC (648) unit, and use the `DebugServerID` as `ServerID` identification. This constant, as well as the record containing the message which is sent between client and server is defined in the msgintf unit.

The dbugintf unit relies on the SimpleIPC (648) mechanism to communicate with the debug server, hence it works on all platforms that have a functional version of that unit. It also uses `TProcess` to

start the debug server if needed, so the process (648) unit should also be functional.

17.4 Constants, types and variables

17.4.1 Resource strings

```
SEntering = '> Entering '
```

String used when sending method enter message.

```
SExiting = '< Exiting '
```

String used when sending method exit message.

```
SProcessID = '%d Process %s (PID=%d)'
```

String used when sending identification message to the server.

```
SSeparator = '>-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-<'
```

String used when sending a separator line.

```
SServerStartFailed = 'Failed to start debugserver. (%s)'
```

String used to display an error message when the start of the debug server failed.

17.4.2 Types

```
TDebugLevel = (dlInformation,dlWarning,dlError)
```

Table 17.2: Enumeration values for type TDebugLevel

Value	Explanation
dlError	Error message.
dlInformation	Informational message.
dlWarning	Warning message.

`TDebugLevel` indicates the severity level of the debug message to be sent. By default, an informational message is sent.

```
TErrorLevel = Array[TDebugLevel] of Integer
```

`TErrorLevel` is used to easily convert an error level enumerated value to an integer value.

17.4.3 Variables

```
DebugServerExe : string = ''
```

`DefaultDebugServerExe` is the filename for the default debug server executable.

```
DefaultDebugServer : string = DebugServerID
```

`DefaultDebugServer` is the name at which the default debug server can be reached.

```
SendError : string = ''
```

Whenever a call encounters an exception, the exception message is stored in this variable.

17.5 Procedures and functions

17.5.1 FreeDebugClient

Synopsis:

Declaration:
```
procedure FreeDebugClient
```

Visibility: default

Description:

17.5.2 GetDebuggingEnabled

Synopsis: Check if sending of debug messages is enabled.

Declaration:
```
function GetDebuggingEnabled : Boolean
```

Visibility: default

Description: `GetDebuggingEnabled` returns the value set by the last call to `SetDebuggingEnabled`. It is `True` by default.

See also: SetDebuggingEnabled (654), SendDebug (651)

17.5.3 InitDebugClient

Synopsis: Initialize the debug client.

Declaration:
```
function InitDebugClient : Boolean
function InitDebugClient(const ShowPID: Boolean;
                         const ServerLogFilename: string) : Boolean
```

Visibility: default

Description: `InitDebugClient` starts the debug server and then performs all necessary initialization of the debug IPC communication channel.

Normally this function should not be called. The SendDebug (651) call will initialize the debug client when it is first called.

Errors: None.

See also: SendDebug (651), StartDebugServer (654)

17.5.4 SendBoolean

Synopsis: Send the value of a boolean variable.

Declaration: `procedure SendBoolean(const Identifier: string; const Value: Boolean)`

Visibility: default

Description: `SendBoolean` is a simple wrapper around SendDebug (651) which sends the name and value of a boolean value as an informational message.

Errors: None.

See also: SendDebug (651), SendDateTime (651), SendInteger (652), SendPointer (653)

17.5.5 SendDateTime

Synopsis: Send the value of a `TDateTime` variable.

Declaration: `procedure SendDateTime(const Identifier: string; const Value: TDateTime)`

Visibility: default

Description: `SendDateTime` is a simple wrapper around SendDebug (651) which sends the name and value of an integer value as an informational message. The value is converted to a string using the DateTimeToStr (**??**) call.

Errors: None.

See also: SendDebug (651), SendBoolean (651), SendInteger (652), SendPointer (653)

17.5.6 SendDebug

Synopsis: Send a message to the debug server.

Declaration: `procedure SendDebug(const Msg: string)`

Visibility: default

Description: `SendDebug` sends the message `Msg` to the debug server as an informational message (debug level `dlInformation`). If no debug server is running, then an attempt will be made to start the server first.

The binary that is started is called debugserver and should be somewhere on the `PATH`. A sample binary which writes received messages to standard output is included in the FCL, it is called dbugsrv. This binary can be renamed to debugserver or can be started before the program is started.

Errors: Errors are silently ignored, any exception messages are stored in SendError (650).

See also: SendDebugEx (651), SendDebugFmt (652), SendDebugFmtEx (652)

17.5.7 SendDebugEx

Synopsis: Send debug message other than informational messages.

Declaration: `procedure SendDebugEx(const Msg: string; MType: TDebugLevel)`

Visibility: default

Description: `SendDebugEx` allows to specify the debug level of the message to be sent in `MType`. By default, SendDebug (651) uses informational messages.

Other than that the function of `SendDebugEx` is equal to that of `SendDebug`

Errors: None.

See also: SendDebug (651), SendDebugFmt (652), SendDebugFmtEx (652)

17.5.8 SendDebugFmt

Synopsis: Format and send a debug message.

Declaration:
```
procedure SendDebugFmt(const Msg: string; const Args: Array of const)
```

Visibility: default

Description: `SendDebugFmt` is a utility routine which formats a message by passing `Msg` and `Args` to Format (**??**) and sends the result to the debug server using SendDebug (651). It exists mainly to avoid the `Format` call in calling code.

Errors: None.

See also: SendDebug (651), SendDebugEx (651), SendDebugFmtEx (652), #rtl.sysutils.format (**??**)

17.5.9 SendDebugFmtEx

Synopsis: Format and send message with alternate type.

Declaration:
```
procedure SendDebugFmtEx(const Msg: string; const Args: Array of const;
                         MType: TDebugLevel)
```

Visibility: default

Description: `SendDebugFmtEx` is a utility routine which formats a message by passing `Msg` and `Args` to Format (**??**) and sends the result to the debug server using SendDebugEx (651) with Debug level `MType`. It exists mainly to avoid the `Format` call in calling code.

Errors: None.

See also: SendDebug (651), SendDebugEx (651), SendDebugFmt (652), #rtl.sysutils.format (**??**)

17.5.10 SendInteger

Synopsis: Send the value of an integer variable.

Declaration:
```
procedure SendInteger(const Identifier: string; const Value: Integer;
                      HexNotation: Boolean)
```

Visibility: default

Description: `SendInteger` is a simple wrapper around SendDebug (651) which sends the name and value of an integer value as an informational message. If `HexNotation` is `True`, then the value will be displayed using hexadecimal notation.

Errors: None.

See also: SendDebug (651), SendBoolean (651), SendDateTime (651), SendPointer (653)

17.5.11 SendMethodEnter

Synopsis: Send method enter message.

Declaration: `procedure SendMethodEnter(const MethodName: string)`

Visibility: default

Description: `SendMethodEnter` sends a "Entering `MethodName`" message to the debug server. After that it increases the message indentation (currently 2 characters). By sending a corresponding SendMethodExit (653), the indentation of messages can be decreased again.

By using the `SendMethodEnter` and `SendMethodExit` methods at the beginning and end of a procedure/method, it is possible to visually trace program execution.

Errors: None.

See also: SendDebug (651), SendMethodExit (653), SendSeparator (654)

17.5.12 SendMethodExit

Synopsis: Send method exit message.

Declaration: `procedure SendMethodExit(const MethodName: string)`

Visibility: default

Description: `SendMethodExit` sends a "Exiting `MethodName`" message to the debug server. After that it decreases the message indentation (currently 2 characters). By sending a corresponding SendMethodEnter (653), the indentation of messages can be increased again.

By using the `SendMethodEnter` and `SendMethodExit` methods at the beginning and end of a procedure/method, it is possible to visually trace program execution.

Note that the indentation level will not be made negative.

Errors: None.

See also: SendDebug (651), SendMethodEnter (653), SendSeparator (654)

17.5.13 SendPointer

Synopsis: Send the value of a pointer variable.

Declaration: `procedure SendPointer(const Identifier: string; const Value: Pointer)`

Visibility: default

Description: `SendInteger` is a simple wrapper around SendDebug (651) which sends the name and value of a pointer value as an informational message. The pointer value is displayed using hexadecimal notation.

Errors: None.

See also: SendDebug (651), SendBoolean (651), SendDateTime (651), SendInteger (652)

17.5.14 SendSeparator

Synopsis: Send a separator message.

Declaration: `procedure SendSeparator`

Visibility: default

Description: `SendSeparator` is a simple wrapper around SendDebug (651) which sends a short horizontal line to the debug server. It can be used to visually separate execution of blocks of code or blocks of values.

Errors: None.

See also: SendDebug (651), SendMethodEnter (653), SendMethodExit (653)

17.5.15 SetDebuggingEnabled

Synopsis: Temporary enables or disables debugging.

Declaration: `procedure SetDebuggingEnabled(const AValue: Boolean)`

Visibility: default

Description: `SetDebuggingEnabled` can be used to temporarily enable or disable sending of debug messages: this allows to control the amount of messages sent to the debug server without having to remove the SendDebug (651) statements. By default, debugging is enabled. If set to false, debug messages are simply discarded till debugging is enabled again.

A value of `True` enables sending of debug messages. A value of `False` disables sending.

Errors: None.

See also: GetDebuggingEnabled (650), SendDebug (651)

17.5.16 StartDebugServer

Synopsis: Start the debug server.

Declaration: `function StartDebugServer(const aLogFilename: string) : Integer`

Visibility: default

Description: `StartDebugServer` attempts to start the debug server. The process started is called debugserver and should be located in the `PATH`.

Normally this function should not be called. The SendDebug (651) call will attempt to start the server by itself if it is not yet running.

Errors: On error, `False` is returned.

See also: SendDebug (651), InitDebugClient (650)

Chapter 18

Reference for unit 'dbugmsg'

18.1 Used units

Table 18.1: Used units by unit 'dbugmsg'

Name	Page
Classes	??
System	??

18.2 Overview

dbugmsg is an auxiliary unit used in the dbugintf (648) unit. It defines the message protocol used between the debug unit and the debug server.

18.3 Constants, types and variables

18.3.1 Constants

```
DebugServerID = 'fpcdebugserver'
```

`DebugServerID` is a string which is used when creating the message protocol, it is used when identifying the server in the (platform dependent) client-server protocol.

```
lctError = 2
```

`lctError` is the identification of error messages.

```
lctIdentify = 3
```

`lctIdentify` is sent by the client to a server when it first connects. It's the first message, and contains the name of client application.

```
lctInformation = 0
```

`lctInformation` is the identification of informational messages.

`lctStop = - 1`

`lctStop` is sent by the client to a server when it disconnects.

`lctWarning = 1`

`lctWarning` is the identification of warning messages.

18.3.2 Types

18.4 Procedures and functions

18.4.1 DebugMessageName

Synopsis: Return the name of the debug message.

Declaration:
```
function DebugMessageName(msgType: Integer) : string
```

Visibility: default

Description: `DebugMessageName` returns the name of the message type. It can be used to examine the `MsgType` field of a TDebugMessage (657) record, and if `msgType` contains a known type, it returns a string describing this type.

Errors: If `MsgType` contains an unknown type, `'Unknown'` is returned.

18.4.2 ReadDebugMessageFromStream

Synopsis: Read a message from stream.

Declaration:
```
procedure ReadDebugMessageFromStream(AStream: TStream;
                                     var Msg: TDebugMessage)
```

Visibility: default

Description: `ReadDebugMessageFromStream` reads a TDebugMessage (657) record (`Msg`) from the stream `AStream`.

The record is not read in a byte-ordering safe way, i.e. it cannot be exchanged between little- and big-endian systems.

Errors: If the stream contains not enough bytes or is malformed, then an exception may be raised.

See also: TDebugMessage (657), WriteDebugMessageToStream (656)

18.4.3 WriteDebugMessageToStream

Synopsis: Write a message to stream.

Declaration:
```
procedure WriteDebugMessageToStream(AStream: TStream;
                                    const Msg: TDebugMessage)
```

Visibility: default

Description: `WriteDebugMessageFromStream` writes a TDebugMessage (657) record (`Msg`) to the stream `AStream`.

The record is not written in a byte-ordering safe way, i.e. it cannot be exchanged between little- and big-endian systems.

Errors: A stream write error may occur if the stream cannot be written to.

See also: TDebugMessage (657), ReadDebugMessageFromStream (656)

18.5 TDebugMessage

```
TDebugMessage = record
  MsgType : Integer;
  MsgTimeStamp : TDateTime
  ;
  Msg : string;
end
```

`TDebugMessage` is a record that describes the message passed from the client to the server. It should not be passed directly in shared memory, as the string containing the message is allocated on the heap. Instead, the WriteDebugMessageToStream (656) and ReadDebugMessageFromStream (656) can be used to read or write the message from/to a stream.

Chapter 19

Reference for unit 'eventlog'

19.1 Used units

Table 19.1: Used units by unit 'eventlog'

Name	Page
Classes	??
System	??
sysutils	??

19.2 Overview

The EventLog unit implements the TEventLog (660) component, which is a component that can be used to send log messages to the system log (if it is available) or to a file.

19.3 Constants, types and variables

19.3.1 Resource strings

```
SErrLogFailedMsg = 'Failed to log entry (Error: %s)'
```

Message used to format an error when an error exception is raised.

```
SErrLogOpenStdErr = 'Standard Error not available for logging'
```

Error message if `stdErr` is not open for writing.

```
SErrLogOpenStdOut = 'Standard Output not available for logging'
```

Error message if `stdOut` is not open for writing.

```
SLogCustom = 'Custom (%d)'
```

Custom message formatting string.

```
SLogDebug = 'Debug'
```

Debug message name.

```
SLogError = 'Error'
```

Error message name.

```
SLogInfo = 'Info'
```

Informational message name.

```
SLogWarning = 'Warning'
```

Warning message name.

19.3.2 Constants

```
AllEvents = [Low(TEventType)..High(TEventType)]
```

19.3.3 Types

```
TLogCategoryEvent = procedure(Sender: TObject; var Code: Word) of
  object
```

`TLogCategoryEvent` is the event type for the TEventLog.OnGetCustomCategory (668) event handler. It should return a OS event category code for the `etCustom` log event type in the `Code` parameter.

```
TLogCodeEvent = procedure(Sender: TObject; var Code: DWord) of
  object
```

`TLogCodeEvent` is the event type for the OnGetCustomEvent (668) and OnGetCustomEventID (668) event handlers. It should return a OS system log code for the `etCustom` log event or event ID type in the `Code` parameter.

```
TLogMessageEvent = procedure(Sender: TObject; EventType: TEventType
  ;
                               const Msg: string) of object
```

`TLogMessageEvent` is the signature of the event handler TEventLog.OnLogMessage (668). If you write your own log message handling method, then it must use this signature.

```
TLogType = (ltSystem,ltFile,ltStdOut,ltStdErr)
```

Table 19.2: Enumeration values for type TLogType

Value	Explanation
ltFile	Write to file.
ltStdErr	Write log messages to standard error output.
ltStdOut	Write log messages to standard output.
ltSystem	Use the system log.

TLogType determines where the log messages are written. It is the type of the TEventLog.LogType (665) property. It can have 2 values:

ltFile This is used to write all messages to file. if no system logging mechanism exists, this is used as a fallback mechanism.

ltSystem This is used to send all messages to the system log mechanism. Which log mechanism this is, depends on the operating system.

19.4 ELogError

19.4.1 Description

ELogError is the exception used in the TEventLog (660) component to indicate errors.

See also: TEventLog (660)

19.5 TEventLog

19.5.1 Description

TEventLog is a component which can be used to send messages to the system log. In case no system log exists (such as on Windows 95/98 or DOS), the messages are written to a file. Messages can be logged using the general Log (663) call, or the specialized Warning (663), Error (664), Info (664) or Debug (664) calls, which have the event type predefined.

See also: Log (663), Warning (663), Error (664), Info (664), Debug (664)

19.5.2 Method overview

Page	Method	Description
661	Create	
664	Debug	Log a debug message.
661	Destroy	Clean up TEventLog instance.
664	Error	Log an error message to.
661	EventTypeToString	Create a string representation of an event type.
664	Info	Log an informational message.
663	Log	Log a message to the system log.
663	Pause	Pause the sending of log messages.
662	RegisterMessageFile	Register message file.
663	Resume	Resume sending of log messages if sending was paused.
662	UnRegisterMessageFile	Unregister the message file (needed on windows only).
663	Warning	Log a warning message.

19.5.3 Property overview

Page	Properties	Access	Description
665	Active	rw	Activate the log mechanism.
664	AppendContent	rw	Control whether output is appended to an existing file.
667	CustomLogType	rw	Custom log type ID.
666	DefaultEventType	rw	Default event type for the Log (663) call.
665	EventFilter	rw	
667	EventIDOffset	rw	Offset for event ID messages identifiers.
666	FileName	rw	File name for log file.
665	Identification	rw	Identification string for messages.
665	LogType	rw	Log type.
668	OnGetCustomCategory	rw	Event to retrieve custom message category.
668	OnGetCustomEvent	rw	Event to retrieve custom event Code.
668	OnGetCustomEventID	rw	Event to retrieve custom event ID.
668	OnLogMessage	rw	Implement custom handling of log messages.
669	Paused	rw	Is the message sending paused ?
666	RaiseExceptionOnError	rw	Determines whether logging errors are reported or ignored.
667	TimeStampFormat	rw	Format for the timestamp string.

19.5.4 TEventLog.Create

Declaration: `constructor Create(aOwner: TComponent);  Override`

Visibility: public

19.5.5 TEventLog.Destroy

Synopsis: Clean up `TEventLog` instance.

Declaration: `destructor Destroy;  Override`

Visibility: public

Description: `Destroy` cleans up the `TEventLog` instance. It cleans any log structures that might have been set up to perform logging, by setting the Active (665) property to False.

See also: Active (665)

19.5.6 TEventLog.EventTypeToString

Synopsis: Create a string representation of an event type.

Declaration: `function EventTypeToString(E: TEventType) : string`

Visibility: public

Description: `EventTypeToString` converts the event type `E` to a suitable string representation for logging purposes. It's mainly used when writing messages to file, as the system log usually has it's own mechanisms for displaying the various event types.

See also: #rtl.sysutils.TEventType (**??**)

19.5.7 TEventLog.RegisterMessageFile

Synopsis: Register message file.

Declaration: `function RegisterMessageFile(AFileName: string) : Boolean;  Virtual`

Visibility: public

Description: `RegisterMessageFile` is used on Windows to register the file `AFileName` containing the formatting strings for the system messages. This should be a file containing resource strings. If `AFileName` is empty, the filename of the application binary is substituted.

When a message is logged to the windows system log, Windows looks for a formatting string in the file registered with this call.

There are 2 kinds of formatting strings:

Category strings these should be numbered from 1 to 4

1 Should contain the description of the `etInfo` event type.

2 Should contain the description of the `etWarning` event type.

4 Should contain the description of the `etError` event type.

4 Should contain the description of the `etDebug` event type.

None of these strings should have a string substitution placeholder.

The second type of strings are the **message definitions**. Their number starts at EventIDOffset (667) (default is 1000) and each string should have 1 placeholder.

Free Pascal comes with a fclel.res resource file which contains default values for the 8 strings, in English. It can be linked in the application binary with the statement

```
{$R fclel.res}
```

This file is generated from the fclel.mc and fclel.rc files that are distributed with the Free Pascal sources.

If the strings are not registered, windows will still display the event messages, but they will not be formatted nicely.

Note that while any messages logged with the event logger are displayed in the event viewer in Windows locks the file registered here. This usually means that the binary is locked.

On non-windows operating systems, this call is ignored.

Errors: If `AFileName` is invalid, false is returned.

19.5.8 TEventLog.UnRegisterMessageFile

Synopsis: Unregister the message file (needed on windows only).

Declaration: `function UnRegisterMessageFile : Boolean;  Virtual`

Visibility: public

Description: `UnRegisterMessageFile` can be used to unregister a message file previously registered with TEventLog.RegisterMessageFile (662). This function is usable only on windows, it has no effect on other platforms. Note that windows locks the registered message file while viewing messages, so unregistering helps to avoid file locks while event viewer is open.

See also: TEventLog.RegisterMessageFile (662)

19.5.9 TEventLog.Pause

Synopsis: Pause the sending of log messages.

Declaration:
```
procedure Pause
```

Visibility: public

Description: `Pause` temporarily suspends the sending of log messages. the various log calls will simply eat the log message and return as if the message was sent.

The sending can be resumed by calling Resume (663).

See also: TEventLog.Resume (663), TEventLog.Paused (669)

19.5.10 TEventLog.Resume

Synopsis: Resume sending of log messages if sending was paused.

Declaration:
```
procedure Resume
```

Visibility: public

Description: `Resume` resumes the sending of log messages if sending was paused through Pause (658).

See also: TEventLog.Pause (663), TEventLog.Paused (669)

19.5.11 TEventLog.Log

Synopsis: Log a message to the system log.

Declaration:
```
procedure Log(EventType: TEventType; const Msg: string);  Virtual
procedure Log(EventType: TEventType; const Fmt: string;
              Args: Array of const)
procedure Log(const Msg: string)
procedure Log(const Fmt: string; Args: Array of const)
```

Visibility: public

Description: `Log` sends a log message to the system log. The message is either the parameter `Msg` as is, or is formatted from the `Fmt` and `Args` parameters. If `EventType` is specified, then it is used as the message event type. If `EventType` is omitted, then the event type is determined from DefaultEventType (666).

If `EventType` is `etCustom`, then the OnGetCustomEvent (668), OnGetCustomEventID (668) and OnGetCustomCategory (668).

The other logging calls: Info (664), Warning (663), Error (664) and Debug (664) use the `Log` call to do the actual work.

See also: Info (664), Warning (663), Error (664), Debug (664), OnGetCustomEvent (668), OnGetCustomEventID (668), OnGetCustomCategory (668)

19.5.12 TEventLog.Warning

Synopsis: Log a warning message.

Declaration:
```
procedure Warning(const Msg: string)
procedure Warning(const Fmt: string; Args: Array of const)
```

Visibility: public

Description: `Warning` is a utility function which logs a message with the `etWarning` type. The message is either the parameter `Msg` as is, or is formatted from the `Fmt` and `Args` parameters.

See also: Log (663), Info (664), Error (664), Debug (664)

19.5.13 TEventLog.Error

Synopsis: Log an error message to.

Declaration:
```
procedure Error(const Msg: string)
procedure Error(const Fmt: string; Args: Array of const)
```

Visibility: public

Description: `Error` is a utility function which logs a message with the `etError` type. The message is either the parameter `Msg` as is, or is formatted from the `Fmt` and `Args` parameters.

See also: Log (663), Info (664), Warning (663), Debug (664)

19.5.14 TEventLog.Debug

Synopsis: Log a debug message.

Declaration:
```
procedure Debug(const Msg: string)
procedure Debug(const Fmt: string; Args: Array of const)
```

Visibility: public

Description: `Debug` is a utility function which logs a message with the `etDebug` type. The message is either the parameter `Msg` as is, or is formatted from the `Fmt` and `Args` parameters.

See also: Log (663), Info (664), Warning (663), Error (664)

19.5.15 TEventLog.Info

Synopsis: Log an informational message.

Declaration:
```
procedure Info(const Msg: string)
procedure Info(const Fmt: string; Args: Array of const)
```

Visibility: public

Description: `Info` is a utility function which logs a message with the `etInfo` type. The message is either the parameter `Msg` as is, or is formatted from the `Fmt` and `Args` parameters.

See also: Log (663), Warning (663), Error (664), Debug (664)

19.5.16 TEventLog.AppendContent

Synopsis: Control whether output is appended to an existing file.

Declaration:
```
Property AppendContent : Boolean
```

Visibility: published

Access: Read,Write

Description: `AppendContent` determines what is done when the log type is `ltFile` and a log file already exists. If the log file already exists, then the default behaviour (`AppendContent=False`) is to re-create the log file when the log is activated. If `AppendContent` is `True` then output will be appended to the existing file.

See also: LogType (665), FileName (666)

19.5.17 TEventLog.Identification

Synopsis: Identification string for messages.

Declaration: `Property Identification : string`

Visibility: published

Access: Read,Write

Description: `Identification` is used as a string identifying the source of the messages in the system log. If it is empty, the filename part of the application binary is used.

See also: Active (665), TimeStampFormat (667)

19.5.18 TEventLog.EventFilter

Declaration: `Property EventFilter : TEventTypes`

Visibility: published

Access: Read,Write

19.5.19 TEventLog.LogType

Synopsis: Log type.

Declaration: `Property LogType : TLogType`

Visibility: published

Access: Read,Write

Description: `LogType` is the type of the log: if it is `ltSystem`, then the system log is used, if it is available. If it is `ltFile` or there is no system log available, then the log messages are written to a file. The name for the log file is taken from the FileName (666) property.

See also: FileName (666)

19.5.20 TEventLog.Active

Synopsis: Activate the log mechanism.

Declaration: `Property Active : Boolean`

Visibility: published

Access: Read,Write

Description: `Active` determines whether the log mechanism is active: if set to `True`, the component connects to the system log mechanism, or opens the log file if needed. Any attempt to log a message while the log is not active will try to set this property to `True`. Disconnecting from the system log or closing the log file is done by setting the `Active` property to `False`.

If the connection to the system logger fails, or the log file cannot be opened, then setting this property may result in an exception.

See also: Log (663)

19.5.21 TEventLog.RaiseExceptionOnError

Synopsis: Determines whether logging errors are reported or ignored.

Declaration: `Property RaiseExceptionOnError : Boolean`

Visibility: published

Access: Read,Write

Description: `RaiseExceptionOnError` determines whether an error during a logging operation will be signaled with an exception or not. If set to `False`, errors will be silently ignored, thus not disturbing normal operation of the program.

19.5.22 TEventLog.DefaultEventType

Synopsis: Default event type for the Log (663) call.

Declaration: `Property DefaultEventType : TEventType`

Visibility: published

Access: Read,Write

Description: `DefaultEventType` is the event type used by the Log (663) call if it's `EventType` parameter is omitted.

See also: Log (663)

19.5.23 TEventLog.FileName

Synopsis: File name for log file.

Declaration: `Property FileName : string`

Visibility: published

Access: Read,Write

Description: `FileName` is the name of the log file used to log messages if no system logger is available or the LogType (665) is `ltFile`. If none is specified, then the name of the application binary is used, with the extension replaced by .log. The file is then located in the /tmp directory on UNIX-like systems, or in the application directory for Dos/Windows like systems.

See also: LogType (665)

19.5.24 TEventLog.TimeStampFormat

Synopsis: Format for the timestamp string.

Declaration: `Property TimeStampFormat : string`

Visibility: published

Access: Read,Write

Description: `TimeStampFormat` is the formatting string used to create a timestamp string when writing log messages to file. It should have a format suitable for the FormatDateTime (**??**) call. If it is left empty, then `yyyy-mm-dd hh:nn:ss.zzz` is used.

See also: TEventLog.Identification (665)

19.5.25 TEventLog.CustomLogType

Synopsis: Custom log type ID.

Declaration: `Property CustomLogType : Word`

Visibility: published

Access: Read,Write

Description: `CustomLogType` is used in the EventTypeToString (661) to format the custom log event type string.

See also: EventTypeToString (661)

19.5.26 TEventLog.EventIDOffset

Synopsis: Offset for event ID messages identifiers.

Declaration: `Property EventIDOffset : DWord`

Visibility: published

Access: Read,Write

Description: `EventIDOffset` is the offset for the message formatting strings in the windows resource file. This property is ignored on other platforms.

The message strings in the file registered with the RegisterMessageFile (662) call are windows resource strings. They each have a unique ID, which must be communicated to windows. In the resource file distributed by Free Pascal, the resource strings are numbered from 1000 to 1004. The actual number communicated to windows is formed by adding the ordinal value of the message's eventtype to `EventIDOffset` (which is by default 1000), which means that by default, the string numbers are:

1000 Custom event types

1001 Information event type

1002 Warning event type

1003 Error event type

1004 Debug event type

See also: RegisterMessageFile (662)

19.5.27 TEventLog.OnGetCustomCategory

Synopsis: Event to retrieve custom message category.

Declaration: `Property OnGetCustomCategory : TLogCategoryEvent`

Visibility: published

Access: Read,Write

Description: `OnGetCustomCategory` is called on the windows platform to determine the category of a custom event type. It should return an ID which will be used by windows to look up the string which describes the message category in the file containing the resource strings.

See also: OnGetCustomEventID (668), OnGetCustomEvent (668)

19.5.28 TEventLog.OnGetCustomEventID

Synopsis: Event to retrieve custom event ID.

Declaration: `Property OnGetCustomEventID : TLogCodeEvent`

Visibility: published

Access: Read,Write

Description: `OnGetCustomEventID` is called on the windows platform to determine the category of a custom event type. It should return an ID which will be used by windows to look up the string which formats the message, in the file containing the resource strings.

See also: OnGetCustomCategory (668), OnGetCustomEvent (668)

19.5.29 TEventLog.OnGetCustomEvent

Synopsis: Event to retrieve custom event Code.

Declaration: `Property OnGetCustomEvent : TLogCodeEvent`

Visibility: published

Access: Read,Write

Description: `OnGetCustomEvent` is called on the windows platform to determine the event code of a custom event type. It should return an ID.

See also: OnGetCustomCategory (668), OnGetCustomEventID (668)

19.5.30 TEventLog.OnLogMessage

Synopsis: Implement custom handling of log messages.

Declaration: `Property OnLogMessage : TLogMessageEvent`

Visibility: published

Access: Read,Write

Description: `OnLogMessage` can be used to implement custom handling of log messages. It is always called, regardless of the LogType (665) setting.

See also: LogType (665), TLogMessageEvent (659)

19.5.31 TEventLog.Paused

Synopsis: Is the message sending paused ?

Declaration: `Property Paused : Boolean`

Visibility: published

Access: Read,Write

Description: `Paused` indicates whether the sending of messages is temporarily suspended or not. Setting it to `True` has the same effect as calling Pause (663). Setting it to `False` has the same effect as calling Resume (663).

See also: TEventLog.Pause (663), TEventLog.Resume (663)

Chapter 20

Reference for unit 'ezcgi'

20.1 Used units

Table 20.1: Used units by unit 'ezcgi'

Name	Page
Classes	??
System	??
sysutils	??

20.2 Overview

ezcgi, written by Michael Hess, provides a single class which offers simple access to the CGI environment which a CGI program operates under. It supports both GET and POST methods. It's intended for simple CGI programs which do not need full-blown CGI support. File uploads are not supported by this component.

To use the unit, a descendent of the `TEZCGI` class should be created and the DoPost (673) or DoGet (673) methods should be overridden.

20.3 Constants, types and variables

20.3.1 Constants

```
hexTable = '0123456789ABCDEF'
```

String constant used to convert a number to a hexadecimal code or back.

20.4 ECGIException

20.4.1 Description

Exception raised by TEZcgi (671).

See also: TEZcgi (671)

20.5 TEZcgi

20.5.1 Description

TEZcgi implements all functionality to analyze the CGI environment and query the variables present in it. It's main use is the exposed variables.

Programs wishing to use this class should make a descendent class of this class and override the DoPost (673) or DoGet (673) methods. To run the program, an instance of this class must be created, and it's Run (672) method should be invoked. This will analyze the environment and call the DoPost or DoGet method, depending on what HTTP method was used to invoke the program.

20.5.2 Method overview

Page	Method	Description
671	Create	Creates a new instance of the TEZCGI component.
671	Destroy	Removes the TEZCGI component from memory.
673	DoGet	Method to handle GET requests.
673	DoPost	Method to handle POST requests.
673	GetValue	Return the value of a request variable.
672	PutLine	Send a line of output to the web-client.
672	Run	Run the CGI application.
672	WriteContent	Writes the content type to standard output.

20.5.3 Property overview

Page	Properties	Access	Description
675	Email	rw	Email of the server administrator.
675	Name	rw	Name of the server administrator.
674	Names	r	Indexed array with available variable names.
673	Values	r	Variables passed to the CGI script.
675	VariableCount	r	Number of available variables.
674	Variables	r	Indexed array with variables as name=value pairs.

20.5.4 TEZcgi.Create

Synopsis: Creates a new instance of the TEZCGI component.

Declaration:
```
constructor Create
```

Visibility: public

Description: Create initializes the CGI program's environment: it reads the environment variables passed to the CGI program and stores them in the Variable (674) property.

See also: Variables (674), Names (674), Values (673)

20.5.5 TEZcgi.Destroy

Synopsis: Removes the TEZCGI component from memory.

Declaration:
```
destructor Destroy;  Override
```

Visibility: public

Description: `Destroy` removes all variables from memory and then calls the inherited destroy, removing the `TEZCGI` instance from memory.

`Destroy` should never be called directly. Instead `Free` should be used, or `FreeAndNil`

See also: Create (671)

20.5.6 TEZcgi.Run

Synopsis: Run the CGI application.

Declaration: `procedure Run`

Visibility: public

Description: `Run` analyses the variables passed to the application, processes the request variables (it stores them in the Variables (674) property) and calls the DoPost (673) or DoGet (673) methods, depending on the method passed to the web server.

After creating the instance of `TEZCGI`, the `Run` method is the only method that should be called when using this component.

See also: Variables (674), DoPost (673), DoGet (673)

20.5.7 TEZcgi.WriteContent

Synopsis: Writes the content type to standard output.

Declaration: `procedure WriteContent(ctype: string)`

Visibility: public

Description: `WriteContent` writes the content type `cType` to standard output, followed by an empty line. After this method was called, no more HTTP headers may be written to standard output. Any HTTP headers should be written before `WriteContent` is called. It should be called from the DoPost (673) or DoGet (673) methods.

See also: DoPost (673), DoGet (673), PutLine (672)

20.5.8 TEZcgi.PutLine

Synopsis: Send a line of output to the web-client.

Declaration: `procedure PutLine(sOut: string)`

Visibility: public

Description: `PutLine` writes a line of text (`sOut`) to the web client (currently, to standard output). It should be called only after WriteContent (672) was called with a content type of `text`. The sent text is not processed in any way, i.e. no HTML entities or so are inserted instead of special HTML characters. This should be done by the user.

Errors: No check is performed whether the content type is right.

See also: WriteContent (672)

20.5.9 TEZcgi.GetValue

Synopsis: Return the value of a request variable.

Declaration: `function GetValue(Index: string; defaultValue: string) : string`

Visibility: public

Description: `GetValue` returns the value of the variable named `Index`, and returns `DefaultValue` if it is empty or does not exist.

See also: Values (673)

20.5.10 TEZcgi.DoPost

Synopsis: Method to handle `POST` requests.

Declaration: `procedure DoPost;  Virtual`

Visibility: public

Description: `DoPost` is called by the Run (672) method the `POST` method was used to invoke the CGI application. It should be overridden in descendents of `TEZcgi` to actually handle the request.

See also: Run (672), DoGet (673)

20.5.11 TEZcgi.DoGet

Synopsis: Method to handle `GET` requests.

Declaration: `procedure DoGet;  Virtual`

Visibility: public

Description: `DoGet` is called by the Run (672) method the `GET` method was used to invoke the CGI application. It should be overridden in descendents of `TEZcgi` to actually handle the request.

See also: Run (672), DoPost (673)

20.5.12 TEZcgi.Values

Synopsis: Variables passed to the CGI script.

Declaration: `Property Values[Index: string]: string`

Visibility: public

Access: Read

Description: `Values` is a name-based array of variables that were passed to the script by the web server or the HTTP request. The `Index` variable is the name of the variable whose value should be retrieved. The following standard values are available:

AUTH_TYPE Authorization type

CONTENT_LENGTH Content length

CONTENT_TYPE Content type

GATEWAY_INTERFACE Used gateway interface

PATH_INFO Requested URL

PATH_TRANSLATED Transformed URL

QUERY_STRING Client query string

REMOTE_ADDR Address of remote client

REMOTE_HOST DNS name of remote client

REMOTE_IDENT Remote identity.

REMOTE_USER Remote user

REQUEST_METHOD Request methods (POST or GET)

SCRIPT_NAME Script name

SERVER_NAME Server host name

SERVER_PORT Server port

SERVER_PROTOCOL Server protocol

SERVER_SOFTWARE Web server software

HTTP_ACCEPT Accepted responses

HTTP_ACCEPT_CHARSET Accepted character sets

HTTP_ACCEPT_ENCODING Accepted encodings

HTTP_IF_MODIFIED_SINCE Proxy information

HTTP_REFERER Referring page

HTTP_USER_AGENT Client software name

Other than the standard list, any variables that were passed by the web-client request, are also available. Note that the variables are case insensitive.

See also: TEZCGI.Variables (674), TEZCGI.Names (674), TEZCGI.GetValue (673), TEZcgi.VariableCount (675)

20.5.13 TEZcgi.Names

Synopsis: Indexed array with available variable names.

Declaration: `Property Names[Index: Integer]: string`

Visibility: public

Access: Read

Description: `Names` provides indexed access to the available variable names. The `Index` may run from 0 to VariableCount (675). Any other value will result in an exception being raised.

See also: TEZcgi.Variables (674), TEZcgi.Values (673), TEZcgi.GetValue (673), TEZcgi.VariableCount (675)

20.5.14 TEZcgi.Variables

Synopsis: Indexed array with variables as name=value pairs.

Declaration: `Property Variables[Index: Integer]: string`

Visibility: public

Access: Read

Description: `Variables` provides indexed access to the available variable names and values. The variables are returned as `Name=Value` pairs. The `Index` may run from 0 to VariableCount (675). Any other value will result in an exception being raised.

See also: TEZcgi.Names (674), TEZcgi.Values (673), TEZcgi.GetValue (673), TEZcgi.VariableCount (675)

20.5.15 TEZcgi.VariableCount

Synopsis: Number of available variables.

Declaration: `Property VariableCount : Integer`

Visibility: public

Access: Read

Description: `TEZcgi.VariableCount` returns the number of available CGI variables. This includes both the standard CGI environment variables and the request variables. The actual names and values can be retrieved with the Names (674) and Variables (674) properties.

See also: Names (674), Variables (674), TEZcgi.Values (673), TEZcgi.GetValue (673)

20.5.16 TEZcgi.Name

Synopsis: Name of the server administrator.

Declaration: `Property Name : string`

Visibility: public

Access: Read,Write

Description: `Name` is used when displaying an error message to the user. This should set prior to calling the TEZcgi.Run (672) method.

See also: TEZcgi.Run (672), TEZcgi.Email (675)

20.5.17 TEZcgi.Email

Synopsis: Email of the server administrator.

Declaration: `Property Email : string`

Visibility: public

Access: Read,Write

Description: `Email` is used when displaying an error message to the user. This should set prior to calling the TEZcgi.Run (672) method.

See also: TEZcgi.Run (672), TEZcgi.Name (675)

Chapter 21

Reference for unit 'FileInfo'

21.1 Used units

Table 21.1: Used units by unit 'FileInfo'

Name	Page
Classes	??
resource	??
System	??
sysutils	??
versionresource	??
versiontypes	??

21.2 Overview

The `FileInfo` unit provides classes and utilities for extracting and manipulating version information from executable files and modules. It includes both low-level access through TVersionInfo (683) and high-level component-based access through TFileVersionInfo (680).

The unit also provides convenience functions for version comparison (CompareVersionQuads (678), CompareProgramVersion (677), NewerVersion (678)), string conversion (VersionQuadToStr (680), ProgramversionToStr (678), StrToVersionQuad (679), StrToProgramVersion (679)), and version extraction (GetProgramVersion (678)).

21.3 Constants, types and variables

21.3.1 Types

```
TVersionCompare = (vcEqual,vcBuildDiffers,vcRevisionDiffers,
  vcMinorDiffers,vcMajorDiffers)
```

Table 21.2: Enumeration values for type TVersionCompare

Value	Explanation
vcBuildDiffers	Only build numbers differ
vcEqual	Versions are identical
vcMajorDiffers	Major version numbers differ
vcMinorDiffers	Minor version numbers or lower differ
vcRevisionDiffers	Revision numbers or lower differ

`TVersionCompare` indicates the level of difference between two version numbers, from equal to major version differences.

```
TVersionQuad = Array[1..4] of Word
```

`TVersionQuad` is an array of four Word values representing major, minor, revision, and build numbers respectively.

21.4 Procedures and functions

21.4.1 assign(TProgramVersion):TVersionQuad

Synopsis: Assignment operator from program version to version quad

Declaration:
```
operator :=(V: TProgramVersion) : TVersionQuad
```

Visibility: default

Description: This operator allows assigning a TProgramVersion (680) to a TVersionQuad (677) variable.

See also: TProgramVersion (680), TVersionQuad (677), assign(TVersionQuad):TProgramVersion (677)

21.4.2 assign(TVersionQuad):TProgramVersion

Synopsis: Assignment operator from version quad to program version

Declaration:
```
operator :=(q: TVersionQuad) : TProgramVersion
```

Visibility: default

Description: This operator allows assigning a TVersionQuad (677) to a TProgramVersion (680) variable.

See also: TVersionQuad (677), TProgramVersion (680), assign(TProgramVersion):TVersionQuad (677)

21.4.3 CompareProgramVersion

Synopsis: Compares two program version records

Declaration:
```
function CompareProgramVersion(Version1: TProgramVersion;
                               Version2: TProgramVersion)
                                : TVersionCompare
```

Visibility: default

Description: `CompareProgramVersion` compares two TProgramVersion (680) records and returns the level of difference between them.

See also: TProgramVersion (680), TVersionCompare (676), CompareVersionQuads (678)

21.4.4 CompareVersionQuads

Synopsis: Compares two version quad arrays

Declaration:
```
function CompareVersionQuads(Quad1: TVersionQuad; Quad2: TVersionQuad)
                                  : TVersionCompare
```

Visibility: default

Description: `CompareVersionQuads` compares two TVersionQuad (677) values and returns the level of difference between them. Comparison is performed hierarchically from major to build version.

See also: TVersionQuad (677), TVersionCompare (676), CompareProgramVersion (677)

21.4.5 GetProgramVersion

Synopsis: Extracts version information from the current executable

Declaration:
```
function GetProgramVersion(out Version: TVersionQuad) : Boolean
function GetProgramVersion(out Version: TProgramVersion) : Boolean
```

Visibility: default

Description: `GetProgramVersion` retrieves the file version from the current executable's version resources. Returns `True` if version information was successfully extracted, `False` otherwise.

See also: TVersionInfo (683)

21.4.6 NewerVersion

Synopsis: Determines if one version is newer than another

Declaration:
```
function NewerVersion(Q1: TVersionQuad; Q2: TVersionQuad) : Boolean
function NewerVersion(V1: TProgramVersion; V2: TProgramVersion)
                        : Boolean
function NewerVersion(V1: string; V2: string) : Boolean
```

Visibility: default

Description: `NewerVersion` compares two version values and returns `True` if the first version is newer than the second. Supports TVersionQuad (677), TProgramVersion (680), and string parameters.

See also: CompareVersionQuads (678), CompareProgramVersion (677)

21.4.7 ProgramVersionToStr

Synopsis: Converts a program version record to string representation

Declaration:
```
function ProgramVersionToStr(const Version: TProgramVersion) : string
```

Visibility: default

Description: `ProgramversionToStr` converts a TProgramVersion (680) to a string in the format "major.minor.revision.build".

See also: TProgramVersion (680), StrToProgramVersion (679), VersionQuadToStr (680)

21.4.8 StrToProgramVersion

Synopsis: Converts a string to a program version record

Declaration:
```
function StrToProgramVersion(const S: string) : TProgramVersion
```

Visibility: default

Description: `StrToProgramVersion` parses a version string and converts it to a TProgramVersion (680). Raises an exception if the string cannot be parsed.

Errors: Raises `EConvertError` if the string is not a valid version format.

See also: TProgramVersion (680), TryStrToProgramVersion (679), StrToVersionQuad (679)

21.4.9 StrToVersionQuad

Synopsis: Converts a string to a version quad

Declaration:
```
function StrToVersionQuad(const S: string) : TVersionQuad
```

Visibility: default

Description: `StrToVersionQuad` parses a version string and converts it to a TVersionQuad (677). Raises an exception if the string cannot be parsed.

Errors: Raises EConvertError if the string is not a valid version quad format.

See also: TVersionQuad (677), TryStrToVersionQuad (679), StrToProgramVersion (679)

21.4.10 TryStrToProgramVersion

Synopsis: Attempts to convert a string to a program version record

Declaration:
```
function TryStrToProgramVersion(const S: string;
                                out Version: TProgramVersion) : Boolean
```

Visibility: default

Description: `TryStrToProgramVersion` parses a version string and converts it to a TProgramVersion (680). Returns `True` if the conversion was successful, `False` otherwise.

See also: TProgramVersion (680), StrToProgramVersion (679), TryStrToVersionQuad (679)

21.4.11 TryStrToVersionQuad

Synopsis: Attempts to convert a string to a version quad

Declaration:
```
function TryStrToVersionQuad(S: string; out Quad: TVersionQuad)
                             : Boolean
```

Visibility: default

Description: `TryStrToVersionQuad` parses a version string and converts it to a TVersionQuad (677). Returns `True` if the conversion was successful, `False` otherwise. Accepts strings in the format "major.minor.revision.build".

See also: TVersionQuad (677), StrToVersionQuad (679), TryStrToProgramVersion (679)

21.4.12 VersionQuadToStr

Synopsis: Converts a version quad to string representation

Declaration: `function VersionQuadToStr(const Quad: TVersionQuad) : string`

Visibility: default

Description: `VersionQuadToStr` converts a TVersionQuad (677) to a string in the format "major.minor.revision.build".

See also: TVersionQuad (677), StrToVersionQuad (679), ProgramversionToStr (678)

21.5 TProgramVersion

```
TProgramVersion = record
  Major : Word;
  Minor : Word;
  Revision
  : Word;
  Build : Word;
end
```

`TProgramVersion` is a record with named fields for major, minor, revision, and build numbers. It provides a more readable alternative to TVersionQuad (677).

21.6 EVersionInfo

21.6.1 Description

`EVersionInfo` is raised when errors occur during version information processing, such as accessing unloaded resources or missing translations.

See also: TVersionInfo (683), TFileVersionInfo (680)

21.7 TFileVersionInfo

21.7.1 Description

`TFileVersionInfo` is a component that provides easy access to version information from executable files. It automatically extracts version strings and supports filtering by key names and translations. The component can be enabled to automatically read version information when properties change.

See also: TVersionInfo (683), EVersionInfo (680)

21.7.2 Method overview

Page	Method	Description
681	Create	Creates a new `TFileVersionInfo` component
681	Destroy	Destroys the `TFileVersionInfo` component
681	Loaded	Called after component loading is complete
681	ReadFileInfo	Reads version information from the specified file

21.7.3 Property overview

Page	Properties	Access	Description
682	Enabled	rw	Controls automatic reading of version information
682	FileName	rw	File name to read version information from
683	Filter	rw	List of version string keys to extract
683	Translation	rw	Language and code page identifier for version strings
683	TranslationOnly	rw	Controls behavior when specified translation is not found
682	VersionStrings	r	Extracted version information strings

21.7.4 TFileVersionInfo.Create

Synopsis: Creates a new `TFileVersionInfo` component

Declaration: `constructor Create(AOwner: TComponent);  Override`

Visibility: public

Description: Initializes the component with empty string lists for version strings and filter.

See also: ReadFileInfo (681)

21.7.5 TFileVersionInfo.Destroy

Synopsis: Destroys the `TFileVersionInfo` component

Declaration: `destructor Destroy;  Override`

Visibility: public

Description: Frees the internal string lists and destroys the component.

See also: Create (681)

21.7.6 TFileVersionInfo.Loaded

Synopsis: Called after component loading is complete

Declaration: `procedure Loaded;  Override`

Visibility: public

Description: Automatically triggers version information reading if the component is enabled after being loaded from a stream.

See also: ReadFileInfo (681)

21.7.7 TFileVersionInfo.ReadFileInfo

Synopsis: Reads version information from the specified file

Declaration: `procedure ReadFileInfo`

Visibility: public

Description: `ReadFileInfo` extracts version strings from the file specified by FileName (682). If no file is specified, it reads from the current executable. The extracted strings are filtered according to Filter (683) and stored in VersionStrings (682).

Errors: Raises EVersionInfo (680) if the specified translation is not found and TranslationOnly (683) is `True`.

See also: FileName (682), VersionStrings (682), Filter (683), Translation (683)

21.7.8 TFileVersionInfo.Enabled

Synopsis: Controls automatic reading of version information

Declaration: `Property Enabled : Boolean`

Visibility: published

Access: Read,Write

Description: When `True`, version information is automatically read when properties change. When `False`, ReadFileInfo (681) must be called manually.

See also: ReadFileInfo (681)

21.7.9 TFileVersionInfo.FileName

Synopsis: File name to read version information from

Declaration: `Property FileName : string`

Visibility: published

Access: Read,Write

Description: Specifies the executable file to extract version information from. If empty, version information is read from the current executable.

See also: ReadFileInfo (681)

21.7.10 TFileVersionInfo.VersionStrings

Synopsis: Extracted version information strings

Declaration: `Property VersionStrings : TStrings`

Visibility: published

Access: Read

Description: Contains the version strings extracted from the file in key=value format. The strings are filtered according to Filter (683) settings.

See also: Filter (683), ReadFileInfo (681)

21.7.11 TFileVersionInfo.Filter

Synopsis: List of version string keys to extract

Declaration: `Property Filter : TStrings`

Visibility: published

Access: Read,Write

Description: Contains the names of version string keys to extract. If empty, all available version strings are extracted. Common keys include CompanyName, FileDescription, FileVersion, ProductName, ProductVersion.

See also: VersionStrings (682)

21.7.12 TFileVersionInfo.Translation

Synopsis: Language and code page identifier for version strings

Declaration: `Property Translation : string`

Visibility: published

Access: Read,Write

Description: Specifies which language and code page combination to use when extracting version strings. The format is an 8-character hexadecimal string (e.g., "040904b0" for English US). If empty, the first available translation is used.

See also: TranslationOnly (683)

21.7.13 TFileVersionInfo.TranslationOnly

Synopsis: Controls behavior when specified translation is not found

Declaration: `Property TranslationOnly : Boolean`

Visibility: published

Access: Read,Write

Description: When `True`, an exception is raised if the specified Translation (683) is not found. When `False`, the first available translation is used instead.

See also: Translation (683)

21.8 TVersionInfo

21.8.1 Description

`TVersionInfo` provides direct access to version resources in executable files. It can load version information from files or module instances and exposes the raw version data structures.

See also: TFileVersionInfo (680), EVersionInfo (680)

21.8.2 Method overview

Page	Method	Description
684	Create	Creates a new `TVersionInfo` instance
684	Destroy	Destroys the `TVersionInfo` instance and frees resources
684	Load	Loads version information from a file or module instance

21.8.3 Property overview

Page	Properties	Access	Description
685	FixedInfo	r	Fixed version information structure
685	StringFileInfo	r	String-based version information
685	VarFileInfo	r	Variable version information containing language and code page data

21.8.4 TVersionInfo.Create

Synopsis: Creates a new `TVersionInfo` instance

Declaration:
```
constructor Create
```

Visibility: public

Description: Initializes the `TVersionInfo` object. No version information is loaded until Load (684) is called.

See also: Load (684)

21.8.5 TVersionInfo.Destroy

Synopsis: Destroys the `TVersionInfo` instance and frees resources

Declaration:
```
destructor Destroy;  Override
```

Visibility: public

Description: Frees all loaded version resources and destroys the object.

21.8.6 TVersionInfo.Load

Synopsis: Loads version information from a file or module instance

Declaration:
```
procedure Load(const Instance: THandle);  Overload
procedure Load(const AFileName: string);  Overload
```

Visibility: public

Description: `Load` extracts version resources from the specified source. It can load from a file name or from a module instance handle. After successful loading, version information is available through the properties.

Errors: May raise exceptions if the file cannot be opened or contains invalid version resources.

See also: FixedInfo (685), StringFileInfo (685), VarFileInfo (685)

21.8.7 TVersionInfo.FixedInfo

Synopsis: Fixed version information structure

Declaration: `Property FixedInfo : TVersionFixedInfo`

Visibility: public

Access: Read

Description: Provides access to the fixed version information containing file version, product version, file flags, and other binary version data.

See also: StringFileInfo (685), VarFileInfo (685)

21.8.8 TVersionInfo.StringFileInfo

Synopsis: String-based version information

Declaration: `Property StringFileInfo : TVersionStringFileInfo`

Visibility: public

Access: Read

Description: Provides access to string version information such as company name, file description, file version, and other textual version data. Information is organized by language and code page.

See also: FixedInfo (685), VarFileInfo (685)

21.8.9 TVersionInfo.VarFileInfo

Synopsis: Variable version information containing language and code page data

Declaration: `Property VarFileInfo : TVersionVarFileInfo`

Visibility: public

Access: Read

Description: Provides access to the variable file information containing supported languages and code pages for the version strings.

See also: FixedInfo (685), StringFileInfo (685)

Chapter 22

Reference for unit 'fpexprpars'

22.1 Used units

Table 22.1: Used units by unit 'fpexprpars'

Name	Page
Classes	**??**
Contnrs	284
System	**??**
sysutils	**??**

22.2 Overview

This unit implements an expression parser and evaluator that supports a wide variety of data types, mathematical operations, and built-in functions. It provides the infrastructure for parsing and evaluating mathematical expressions with support for variables, functions, and complex data types.

The parser supports the following features:

- Multiple data types: integers, floats, strings, booleans, dates, and currency
- Standard mathematical operators: +, -, *, /, mod, power
- Comparison operators: =, <>, <, >, <=, >=
- Logical operators: and, or, xor, not
- Built-in mathematical, string, and date/time functions comparable to what exists in the SysUtils (**??**) unit.
- Variable and function definitions
- Conditional expressions (if-then-else)
- Aggregate functions for data processing

The unit is designed to be extensible, allowing custom functions and variables to be registered with the parser for specialized applications.

22.3 Constants, types and variables

22.3.1 Constants

```
AllBuiltIns = [bcStrings, bcDateTime, bcMath, bcBoolean, bcConversion
  , bcData, bcVaria, bcUser, bcAggregate]
```

This constant represents the complete set of all available built-in identifier categories in the expression parser. It includes all predefined categories such as string functions, date/time operations, mathematical functions, boolean operations, data conversion functions, and user-defined functions.

This provides a convenient way to enable the full feature set of the expression parser without having to specify individual categories manually.

```
ttComparisons = [ttLargerThan, ttLessthan, ttLargerThanEqual, ttLessthanEqual
  , ttEqual, ttUnequal]
```

This constant set contains all TTokenType (691) values that represent comparison operators in expressions. It includes equality, inequality, and ordering operators such as equal, unequal, less than, greater than, less than or equal, and greater than or equal.

The set is used by the parser to identify comparison operations and ensure proper precedence handling when evaluating expressions.

```
ttDelimiters = [ttPlus, ttMinus, ttLessThan, ttLargerThan, ttEqual
  , ttDiv, ttMul, ttLeft, ttRight, ttLessThanEqual, ttLargerThanEqual
  , ttunequal, ttPower]
```

This constant set contains all TTokenType (691) values that represent operators and structural delimiters in expressions. It includes arithmetic operators, comparison operators, parentheses, and the power operator.

The set is used by the parser and scanner to identify tokens that serve as operators or structural elements, allowing the parser to distinguish between operands and operators during expression analysis.

22.3.2 Types

```
PFPExpressionResult = ^TFPExpressionResult
```

This type defines a pointer to a TFPExpressionResult (695) record. It is used in function interfaces where results need to be passed by reference or when dynamic allocation of result records is required.

```
TBuiltInCategories = Set of TBuiltInCategory
```

This set type allows for specifying collections of TBuiltInCategory (687) values. It is used to define which categories of built-in functions should be enabled or available in specific contexts.

```
TBuiltInCategory = (bcStrings,bcDateTime,bcMath,bcBoolean,bcConversion
  ,
                    bcData,bcVaria,bcUser,bcAggregate)
```

Table 22.2: Enumeration values for type TBuiltInCategory

Value	Explanation
bcAggregate	Aggregate functions category
bcBoolean	Boolean functions category
bcConversion	Type conversion functions category
bcData	Data manipulation functions category
bcDateTime	Date and time functions category
bcMath	Mathematical functions category
bcStrings	String manipulation functions category
bcUser	User-defined functions category
bcVaria	Miscellaneous functions category

This enumeration categorizes the different types of built-in functions available in the expression parser. Each category groups related functions together, such as string functions, mathematical functions, date/time functions, and others.

```
TExprArgumentArray = Array of TFPExprNode
```

The `TExprArgumentArray` array type holds the argument nodes passed to functions and operations that accept multiple parameters. Each element is a TFPExprNode (750) representing one argument expression.

The array is dynamically sized to accommodate functions with varying numbers of arguments. Functions and operations use this array to access their argument nodes and evaluate them during expression processing.

Unlike TExprParameterArray (688) which holds evaluated results, this array holds the actual expression nodes that produce the argument values.

This class is used internally by the TFPExpressionParser (728) class to construct an AST (Abstract Syntax Tree). There is normally no need for you to use or instantiate this class directly.

```
TExprFloat = Double
```

This type alias defines the standard floating-point data type used throughout the expression parser for all floating-point calculations and storage. It is currently defined as Double, providing the highest cross-platform precision for mathematical operations.

All floating-point operations within the expression parser, including mathematical functions and numeric conversions, use this type to ensure consistency and precision. The type is used in expression results, intermediate calculations, and function parameters.

Using a type alias allows for easy modification of the underlying floating-point precision across the entire expression parsing system if needed for specific applications or platforms.

```
TExprParameterArray = Array of TFPExpressionResult
```

This array type holds the parameter values passed to functions during expression evaluation. Each element is a TFPExpressionResult (695) containing both the value and type information for one parameter.

The array is dynamically sized to accommodate functions with varying numbers of parameters. Functions use this array to access their input parameters and determine their types during evaluation.

```
TFPBinaryOperationClass = Class of TFPBinaryOperation
```

This is a class reference type (metaclass) for TFPBinaryOperation (716). It allows for dynamic creation of binary operation instances and enables polymorphic handling of different binary operation types during expression parsing.

Class reference types are essential in factory patterns where the specific binary operation class to be instantiated is determined at runtime based on the operator token encountered in the expression (such as +, -, *, /, etc.).

```
TFPBooleanResultOperationClass = Class of
  TFPBooleanResultOperation
```

This is a class reference type (metaclass) for TFPBooleanResultOperation (721). It allows for dynamic creation of boolean result operation instances and enables polymorphic handling of different boolean result operation types.

```
TFPExpressionParserClass = Class of TFPExpressionParser
```

This is a class reference type that can hold references to TFPExpressionParser (728) or its derived classes. It enables polymorphic creation and manipulation of parser instances.

This type is useful for applications that need to work with different parser implementations or create parser instances dynamically based on runtime conditions.

```
TFPExprFunctionCallBack = procedure(var Result: TFPExpressionResult
  ;
                                    const Args: TExprParameterArray
  )
```

This procedural type defines the signature for callback procedures that can be used to implement custom function evaluation. It allows external code to provide function implementations that will be called during expression evaluation.

```
TFPExprFunctionClass = Class of TFPExprFunction
```

This is a class reference type (metaclass) for TFPExprFunction (737).

This type is used by the function registry to store references to function classes that can be instantiated when the corresponding function is called in an expression.

```
TFPExprFunctionEvent = procedure(var Result: TFPExpressionResult;
  const Args: TExprParameterArray
  )
                                  of object
```

This event procedure type defines the signature for event handlers that implement custom function evaluation. It provides a method-based approach to function implementation using object methods rather than standalone procedures.

```
TFPExprVariableCallBack = procedure(var Result: TFPExpressionResult
  ;
                                    constref AName: ShortString
  )
```

This procedural type defines the signature for callback procedures that provide variable values during expression evaluation. It allows external code to resolve variable names to their corresponding values at runtime.

```
TFPExprVariableEvent = procedure(var Result: TFPExpressionResult;
  constref AName: ShortString) of
  object
```

This event procedure type defines the signature for event handlers that resolve variable values during expression evaluation. It provides a method-based approach to variable resolution using object methods rather than standalone procedures.

```
TIdentifierEvent = procedure(Sender: TObject;
  const aIdentifier: AnsiString;
  var aIdent: TFPExprIdentifierDef) of object


TIdentifierType = (itVariable,itFunctionCallBack,itFunctionHandler
  ,
                   itFunctionNode)
```

Table 22.3: Enumeration values for type TIdentifierType

Value	Explanation
itFunctionCallBack	Function callback identifier type
itFunctionHandler	Function event handler identifier type
itFunctionNode	Function node identifier type
itVariable	Variable identifier type

This enumeration defines the different types of identifiers that can be encountered in expressions. It distinguishes between variables and functions to enable proper handling and resolution during expression parsing and evaluation.

```
TNumberKind = (nkDecimal,nkHex,nkOctal,nkBinary)
```

Table 22.4: Enumeration values for type TNumberKind

Value	Explanation
nkBinary	Binary number format (base 2)
nkDecimal	Decimal number format (base 10)
nkHex	Hexadecimal number format (base 16)
nkOctal	Octal number format (base 8)

This enumeration defines the different numerical base formats that the expression scanner can recognize and parse when tokenizing numeric literals in expressions.

The scanner can handle numbers in various bases commonly used in programming and mathematical contexts, allowing expressions to contain numbers in the most convenient format for the specific use case.

Each kind represents a different base system with its own parsing rules and digit validation requirements during the scanning process.

```
TResultType = (rtBoolean,rtInteger,rtFloat,rtDateTime,rtString,
  rtCurrency)
```

Table 22.5: Enumeration values for type TResultType

Value	Explanation
rtBoolean	Boolean result type
rtCurrency	Currency result type
rtDateTime	Date/time result type
rtFloat	Floating-point result type
rtInteger	Integer result type
rtString	String result type

This enumeration defines all possible data types that can be produced as results from expression evaluation. Each value represents a specific data type that expression nodes can return, enabling type checking and proper value handling.

The result type system ensures that expressions are properly typed and that operations between different types are handled correctly through appropriate type conversions when needed.

These types correspond to the variant record fields in TFPExpressionResult (695) and determine how result values should be interpreted and stored.

```
TResultTypes = Set of TResultType
```

This set type allows for specifying collections of TResultType (691) values. It is used in validation and type checking to define which result types are acceptable for specific operations or functions.

For example, arithmetic operations might accept a set containing rtInteger and rtFloat types, while comparison operations might accept a broader set of types that can be compared.

```
TTokenType = (ttPlus,ttMinus,ttLessThan,ttLargerThan,ttEqual,ttDiv
  ,
              ttMod,ttMul,ttLeft,ttRight,ttLessThanEqual,
  ttLargerThanEqual,ttunequal,ttNumber,ttString,
  ttIdentifier,ttComma,ttAnd,ttOr,ttXor,ttTrue,ttFalse,
  ttNot,ttif,ttCase,ttPower,ttEOF)
```

Table 22.6: Enumeration values for type TTokenType

Value	Explanation
ttAnd	Logical AND operator token (and)
ttCase	Case conditional keyword token
ttComma	Comma separator token (,)
ttDiv	Division operator token (/)
ttEOF	End of file token marking end of input
ttEqual	Equality comparison operator token (=)
ttFalse	Boolean false literal token
ttIdentifier	Identifier token (variable or function name)
ttif	Conditional if keyword token
ttLargerThan	Greater than comparison operator token (>)
ttLargerThanEqual	Greater than or equal comparison operator token (>=)
ttLeft	Left parenthesis token (()
ttLessThan	Less than comparison operator token (<)
ttLessThanEqual	Less than or equal comparison operator token (<=)
ttMinus	Minus operator token (-)
ttMod	Modulo operator token (mod)
ttMul	Multiplication operator token (*)
ttNot	Logical NOT operator token (not)
ttNumber	Numeric literal token
ttOr	Logical OR operator token (or)
ttPlus	Plus operator token (+)
ttPower	Power/exponentiation operator token (**)
ttRight	Right parenthesis token ())
ttString	String literal token
ttTrue	Boolean true literal token
ttunequal	Inequality comparison operator token (<>)
ttXor	Logical XOR operator token (xor)

This enumeration defines all possible token types that can be encountered during expression parsing and lexical analysis. Each token type represents a fundamental unit of the expression language, from operators and keywords to literals and identifiers.

The token types are organized into several categories:

- Arithmetic operators: ttPlus, ttMinus, ttMul, ttDiv, ttMod, ttPower
- Comparison operators: ttEqual, ttUnequal, ttLessThan, ttLargerThan, ttLessThanEqual, ttLargerThanEqual
- Logical operators: ttAnd, ttOr, ttXor, ttNot
- Literals: ttNumber, ttString, ttTrue, ttFalse
- Structure tokens: ttLeft, ttRight, ttComma
- Keywords: ttIf, ttCase
- Special tokens: ttIdentifier, ttEOF

The TFPExpressionScanner (734) uses these token types to categorize input text during the lexical analysis phase of expression parsing.

22.4 Procedures and functions

22.4.1 ArgToFloat

Synopsis: Convert argument to float value

Declaration: `function ArgToFloat(Arg: TFPExpressionResult) : TExprFloat`

Visibility: default

Description: This utility function converts an TFPExpressionResult (695) argument to a floating-point value. It handles conversion from various result types including integer, currency, and string representations to the standard TExprFloat (688) type.

The function is primarily used internally by mathematical operations and built-in functions that require floating-point operands. It provides type safety by ensuring consistent conversion behavior across different numeric types.

If the argument cannot be converted to a valid floating-point value, appropriate error handling is performed to maintain expression evaluation integrity.

22.4.2 BuiltinIdentifiers

Synopsis: Get built-in identifiers manager

Declaration: `function BuiltinIdentifiers : TExprBuiltInManager`

Visibility: default

Description: The `BuiltinIdentifiers` function provides access to the global TExprBuiltInManager (708) instance that contains all standard built-in functions and variables available to the expression parser. The manager is automatically initialized with default built-in functions when first accessed.

The returned manager includes predefined mathematical functions (sin, cos, sqrt, etc.), string manipulation functions (length, uppercase, etc.), date/time functions (now, date, etc.), and conversion functions between different data types.

This global manager can be extended with custom functions and variables, or individual parsers can use their own manager instances for specialized functionality.

22.4.3 CharToResultType

Synopsis: Convert character to result type

Declaration: `function CharToResultType(C: AnsiChar) : TResultType`

Visibility: default

Description: The `CharToResultType` function converts a single character code to the corresponding TResultType (691) enumeration value. It is used internally for parsing type specifications and managing result type conversions within the expression parser.

The character codes map to specific result types:

- 'B' maps to Boolean type (rtBoolean)
- 'I' maps to Integer type (rtInteger)
- 'F' maps to Float type (rtFloat)
- 'S' maps to String type (rtString)

- 'D' maps to DateTime type (rtDateTime)
- 'C' maps to Currency type (rtCurrency)

The `CharToResultType` function is primarily used for internal type management and serialization purposes within the expression parsing framework.

22.4.4 RegisterStdBuiltins

Synopsis: Register standard built-in functions

Declaration:
```
procedure RegisterStdBuiltins(AManager: TExprBuiltInManager;
                              Categories: TBuiltInCategories)
```

Visibility: default

Description: This procedure registers the standard set of built-in functions and variables with the specified manager. It populates the manager with predefined mathematical, string, date/time, and utility functions based on the selected categories.

The registration process allows applications to customize which built-in functions are available by specifying categories, enabling selective loading of functionality to meet specific requirements while minimizing memory usage.

22.4.5 ResultTypeName

Synopsis: Get name for result type

Declaration:
```
function ResultTypeName(AResult: TResultType) : AnsiString
```

Visibility: default

Description: The ResultTypeName (694) function converts a TResultType (691) enumeration value to its corresponding string name. It provides a human-readable representation of result types for debugging, error reporting, and logging purposes.

22.4.6 TokenName

Synopsis: Get name for token type

Declaration:
```
function TokenName(AToken: TTokenType) : AnsiString
```

Visibility: default

Description: The TokenName (694) function converts a TTokenType (691) enumeration value to its corresponding string name. It provides a human-readable representation of token types for debugging, error reporting, and logging purposes.

22.5 TFPExpressionResult

```
TFPExpressionResult = record
  ResString : AnsiString;
case ResultType
  : TResultType of
rtBoolean: (
```

```
  ResBoolean : Boolean;
);
rtInteger
  : (
  ResInteger : Int64;
);
rtFloat: (
  ResFloat : TExprFloat;
  );
rtCurrency: (
  ResCurrency : Currency;
);
rtDateTime: (
  ResDateTime
  : TDatetime;
);
rtString: (
);
end
```

The `TFPExpressionResult` record stores the result of evaluating an expression node, containing both the actual value and type information.

The `ResultType` field indicates which variant is active and determines how to interpret the value stored in the record.

22.6 EExprParser

22.6.1 Description

This exception class is raised by the TFPExpressionParser (728) when parsing or evaluation errors occur. It encompasses both syntactic errors during expression parsing and runtime errors during expression evaluation.

This exception is raised in various scenarios:

- Syntax errors in expression structure (mismatched parentheses, invalid operators)
- Type mismatches during expression evaluation
- Undefined variable or function references
- Runtime errors during function calls or mathematical operations
- Invalid function arguments or parameter count mismatches
- Division by zero and other mathematical errors

22.7 EExprScanner

22.7.1 Description

This exception class is raised by the TFPExpressionScanner (734) when lexical analysis errors occur during expression scanning. It indicates problems encountered while tokenizing the input expression text.

Common scenarios that trigger this exception include:

- Invalid character sequences that cannot be recognized as valid tokens
- Malformed string literals (unclosed quotes)
- Invalid numeric literals (malformed number formats)
- Unexpected characters that violate the expression grammar

The exception typically provides detailed error information including the position where the error occurred and a description of what went wrong during the scanning process.

22.8 TAggregateAvg

22.8.1 Description

The `TAggregateAvg` class implements the average (mean) aggregate function, which calculates the arithmetic mean of all processed numeric values. It extends TAggregateSum (701) to leverage summation functionality while adding count tracking for average calculation.

The average function handles different numeric types and ensures appropriate precision in the final result. Division by zero is handled appropriately when no values have been processed.

This class is used internally by the TFPExpressionParser (728) class to construct an AST (Abstract Syntax Tree). There is normally no need for you to use or instantiate this class directly.

See also: TFPExprNode (750)

22.8.2 Method overview

Page	Method	Description
697	GetNodeValue	Calculate the average from aggregated values
696	InitAggregate	Initialize average aggregation
697	UpdateAggregate	Update average aggregate with new value

22.8.3 TAggregateAvg.InitAggregate

Synopsis: Initialize average aggregation

Declaration: `procedure InitAggregate;  Override`

Visibility: public

Description: The `InitAggregate` method initializes the average aggregate for a new aggregation cycle. It resets both the sum accumulator and the count of processed values to zero, preparing the aggregate for processing new values.

The method calls the parent TAggregateSum.InitAggregate (702) to handle sum initialization and additionally resets the count tracking needed for average calculation.

22.8.4 TAggregateAvg.UpdateAggregate

Synopsis: Update average aggregate with new value

Declaration: `procedure UpdateAggregate;  Override`

Visibility: public

Description: The `UpdateAggregate` method processes a new value for the average calculation by updating both the running sum and incrementing the count of processed values. It leverages the parent TAggregateSum.UpdateAggregate (702) for sum handling.

Each call to this method adds one value to the sum and increases the count by one, maintaining the data needed to calculate the final average when requested.

22.8.5 TAggregateAvg.GetNodeValue

Synopsis: Calculate the average from aggregated values

Declaration: `procedure GetNodeValue(var Result: TFPExpressionResult);  Override`

Visibility: public

Description: The `GetNodeValue` method calculates and returns the final average value by dividing the accumulated sum by the count of processed values. It overrides the base class behavior to provide average-specific result calculation.

The method handles edge cases such as division by zero (when no values have been processed) and ensures appropriate result type conversion based on the input data types.

The returned result is a TFPExpressionResult (695) containing the calculated average as a floating-point value.

22.9 TAggregateCount

22.9.1 Description

The `TAggregateCount` class implements the count aggregate function, which counts the number of values processed during aggregation. It extends TAggregateExpr (698) to provide count-specific aggregation logic.

Unlike other aggregate functions that process the actual values, the count function simply increments a counter for each value encountered, regardless of the value's content or type. This makes it useful for determining the number of records or values in a dataset.

The count function always returns an integer result representing the total number of values processed during the aggregation cycle.

See also: TFPExprNode (750)

22.9.2 Method overview

Page	Method	Description
698	InitAggregate	Initialize count aggregation
698	UpdateAggregate	Update count aggregate with new value

22.9.3 TAggregateCount.InitAggregate

Synopsis: Initialize count aggregation

Declaration: `procedure InitAggregate;  Override`

Visibility: public

Description: The `InitAggregate` method initializes the count aggregate for a new aggregation cycle. It resets the internal counter to zero, preparing the aggregate to count new values from the beginning.

The method sets up the initial state needed for accurate counting and ensures that the counter starts from zero for each new aggregation sequence.

Proper initialization is essential for accurate count results, as it establishes the starting point for the counting process.

22.9.4 TAggregateCount.UpdateAggregate

Synopsis: Update count aggregate with new value

Declaration: `procedure UpdateAggregate;  Override`

Visibility: public

Description: The `UpdateAggregate` method increments the internal counter by one for each value encountered during the aggregation process. Unlike other aggregates, it does not examine the actual value content, only the presence of a value.

Each call to this method represents the processing of one item in the aggregation sequence, and the counter is incremented regardless of the value's type or content.

This simple counting approach makes the count aggregate efficient and applicable to any data type or value combination.

22.10 TAggregateExpr

22.10.1 Description

This abstract base class provides the foundation for all aggregate expression classes in the expression parser. Aggregate expressions perform calculations over multiple values, such as sums, averages, counts, minimums, and maximums.

The class extends TFPExprNode (750) to provide aggregate-specific functionality including initialization and update methods that are called as values are processed. Derived classes must implement the specific aggregation logic for their particular operation.

Aggregate expressions typically maintain internal state that accumulates as values are processed, making them stateful unlike regular expression nodes. The aggregation process involves initialization, iterative updates, and final value calculation.

This class is used internally by the TFPExpressionParser (728) class to construct an AST (Abstract Syntax Tree). There is normally no need for you to use or instantiate this class directly.

See also: TFPExprNode (750)

22.10.2 Method overview

Page	Method	Description
699	GetNodeValue	Get the aggregate expression value
699	IsAggregate	Check if expression is an aggregate

22.10.3 TAggregateExpr.IsAggregate

Synopsis: Check if expression is an aggregate

Declaration: `class function IsAggregate : Boolean;  Override`

Visibility: public

Description: The `TAggregateExpr.IsAggregate` class function returns True to indicate that this expression is an aggregate expression. It overrides the default behavior from TFPExprNode (750) which returns False for regular expressions.

The method is used by the expression parser to identify aggregate expressions and handle them differently from regular expressions, as aggregates require special processing including initialization and incremental updates.

22.10.4 TAggregateExpr.GetNodeValue

Synopsis: Get the aggregate expression value

Declaration: `procedure GetNodeValue(var Result: TFPExpressionResult);  Override`

Visibility: public

Description: The `GetNodeValue` method returns the final calculated value of the aggregate expression after all values have been processed. It is called after the aggregation process is complete to retrieve the accumulated result.

The method returns the result in a TFPExpressionResult (695) structure, which can contain different data types depending on the specific aggregate operation (numeric for sums and averages, integer for counts, etc.).

The GetNodeValue (699) method should only be called after the aggregation initialization and update phases are complete, as the result depends on all processed values.

22.11 TAggregateMax

22.11.1 Description

The `TAggregateMax` class implements the maximum aggregate function, which finds the largest value among all processed values. It extends TAggregateExpr (698) to provide maximum-specific aggregation logic.

The class maintains an internal state that tracks the maximum value encountered so far during the aggregation process. The first value encountered is used as the initial maximum, and subsequent values are compared to update the maximum when larger values are found.

The maximum function works with numeric types (integers, floats, currency) and can also compare string and date/time values using appropriate comparison logic for each data type.

See also: TFPExprNode (750)

22.11.2 Method overview

Page	Method	Description
700	InitAggregate	Initialize maximum aggregation
700	UpdateAggregate	Update maximum aggregate with new value

22.11.3 TAggregateMax.InitAggregate

Synopsis: Initialize maximum aggregation

Declaration: `procedure InitAggregate;  Override`

Visibility: public

Description: The `InitAggregate` method initializes the maximum aggregate for a new aggregation cycle. It sets up the internal state needed to track the maximum value across multiple value updates.

The method resets an internal flag, indicating that the next value processed will be the first value in the aggregation sequence. It also prepares any internal storage needed for the maximum calculation.

The InitAggregate (700) method must be called before starting a new aggregation cycle to ensure proper initialization of the aggregate state.

22.11.4 TAggregateMax.UpdateAggregate

Synopsis: Update maximum aggregate with new value

Declaration: `procedure UpdateAggregate;  Override`

Visibility: public

Description: The `UpdateAggregate` method processes a new value in the maximum aggregation sequence. If this is the first value, it becomes the initial maximum. Otherwise, the new value is compared with the current maximum and replaces it if larger.

The method handles different data types appropriately, using numeric comparison for numbers and lexical comparison for strings. Type compatibility is maintained throughout the aggregation process.

After processing the first value, the FFirst flag is set to False to ensure subsequent values are properly compared against the established maximum.

22.12 TAggregateMin

22.12.1 Description

The `AggregateMin` class implements the minimum aggregate function, which finds the smallest value among all processed values. It extends TAggregateExpr (698) to provide minimum-specific aggregation logic.

The class maintains an internal state that tracks the minimum value encountered so far during the aggregation process. The first value encountered is used as the initial minimum, and subsequent values are compared to update the minimum when smaller values are found.

The minimum function works with numeric types (integers, floats, currency) and can also compare string and date/time values using appropriate comparison logic for each data type.

This class is used internally by the TFPExpressionParser (728) class to construct an AST (Abstract Syntax Tree). There is normally no need for you to use or instantiate this class directly.

See also: TFPExprNode (750)

22.12.2 Method overview

Page	Method	Description
701	InitAggregate	Initialize minimum aggregation
701	UpdateAggregate	Update minimum aggregate with new value

22.12.3 TAggregateMin.InitAggregate

Synopsis: Initialize minimum aggregation

Declaration: `procedure InitAggregate;  Override`

Visibility: public

Description: The `InitAggregate` method initializes the minimum aggregate for a new aggregation cycle. It sets up the internal state needed to track the minimum value across multiple value updates.

The method sets an internal flag, indicating that the next value processed will be the first value in the aggregation sequence. It also prepares any internal storage needed for the minimum calculation.

The `InitAggregate` method must be called before starting a new aggregation cycle to ensure proper initialization of the aggregate state.

22.12.4 TAggregateMin.UpdateAggregate

Synopsis: Update minimum aggregate with new value

Declaration: `procedure UpdateAggregate;  Override`

Visibility: public

Description: The `UpdateAggregate` method processes a new value in the minimum aggregation sequence. If this is the first value, it becomes the initial minimum. Otherwise, the new value is compared with the current minimum and replaces it if smaller.

After processing the first value, the FFirst flag is set to False to ensure subsequent values are properly compared against the established minimum.

22.13 TAggregateSum

22.13.1 Description

The `TAggregateSum` class implements the sum aggregate function, which calculates the total of all processed numeric values. It extends TAggregateExpr (698) to provide summation-specific aggregation logic.

The class maintains an internal accumulator that adds each new value to the running total. It handles different numeric types (integers, floats, currency) and performs appropriate type conversions to maintain precision during the summation process.

The sum function ensures type safety by converting arguments to compatible numeric types before performing addition operations. The final result type depends on the types of values processed during aggregation.

This class is used internally by the TFPExpressionParser (728) class to construct an AST (Abstract Syntax Tree). There is normally no need for you to use or instantiate this class directly.

See also: TFPExprNode (750)

22.13.2 Method overview

Page	Method	Description
702	ConvertArgument	Convert argument for sum aggregation
702	InitAggregate	Initialize sum aggregation
702	UpdateAggregate	Update sum aggregate with new value

22.13.3 TAggregateSum.ConvertArgument

Synopsis: Convert argument for sum aggregation

Declaration:
```
function ConvertArgument(aIndex: Integer; aNode: TFPExprNode;
                         aType: TResultType) : TFPExprNode;  Override
```

Visibility: public

Description: The `ConvertArgument` method converts an expression node argument to an appropriate numeric type for summation. It handles type conversion between different numeric types to ensure compatibility during the sum calculation process.

The method takes the argument index, the expression node containing the value, and the target result type, then performs the necessary conversion to make the value compatible with the sum operation.

This conversion ensures that mixed numeric types can be properly summed together while maintaining appropriate precision and avoiding type-related runtime errors.

22.13.4 TAggregateSum.InitAggregate

Synopsis: Initialize sum aggregation

Declaration:
```
procedure InitAggregate;  Override
```

Visibility: public

Description: The `InitAggregate` method initializes the sum aggregate for a new aggregation cycle. It resets the internal accumulator to zero and prepares the aggregate state for processing new values.

The method sets up the appropriate initial values based on the expected result type (integer, float, or currency) and ensures that the accumulator is ready to receive and sum new values.

22.13.5 TAggregateSum.UpdateAggregate

Synopsis: Update sum aggregate with new value

Declaration:
```
procedure UpdateAggregate;  Override
```

Visibility: public

Description: The `UpdateAggregate` method adds a new value to the running sum during the aggregation process. It handles the addition of different numeric types while maintaining appropriate precision and type compatibility.

The method performs type conversion as needed using the ConvertArgument (702) method, then adds the converted value to the internal accumulator. It ensures that the sum calculation maintains the appropriate result type throughout the process.

Each call to this method processes one value from the aggregation sequence and updates the internal sum state accordingly.

22.14 TCaseOperation

22.14.1 Description

The `TCaseOperation` class implements case-when conditional operations in expressions, providing multi-way branching logic based on value matching. It allows for testing a single expression against multiple possible values and executing different branches based on which value matches.

The case operation evaluates a test expression and compares it against a series of when-value pairs, executing the associated expression for the first matching value. If no values match, an optional else expression is evaluated.

The `TCaseOperation` class enables complex multi-conditional logic within expressions, similar to CASE statements in SQL or switch statements in programming languages, providing an alternative to nested if-then-else constructs.

This class is used internally by the TFPExpressionParser (728) class to construct an AST (Abstract Syntax Tree). There is normally no need for you to use or instantiate this class directly.

See also: TFPExprNode (750)

22.14.2 Method overview

Page	Method	Description
705	AsString	Get string representation of case operation
703	Check	Validate the case operation for correctness
704	Create	Create case operation with arguments
705	destroy	Destroy case operation and free resources
704	HasAggregate	Check if case operation contains aggregates
703	InitAggregate	Initialize aggregate processing for case operation
704	NodeType	Get the result type for case operation
704	UpdateAggregate	Update aggregate calculations for case operation

22.14.3 Property overview

Page	Properties	Access	Description
705	Condition	r	The condition expression of the case operation

22.14.4 TCaseOperation.Check

Synopsis: Validate the case operation for correctness

Declaration: `procedure Check;  Override`

Visibility: public

Description: The Check (703) method validates the case operation to ensure that all components are properly configured and type-compatible. It checks that the condition expression and all case branches have compatible types and that the operation can be safely evaluated.

The validation process includes verifying that the condition can be evaluated to determine which case branch should be executed, and that all possible execution paths result in compatible data types.

22.14.5 TCaseOperation.InitAggregate

Synopsis: Initialize aggregate processing for case operation

Declaration: `procedure InitAggregate;  Override`

Visibility: public

Description: The `InitAggregate` method initializes aggregate processing for the case operation node. It prepares the case operation for aggregate calculations by initializing any aggregate functions used within the case branches.

The initialization process ensures that all child nodes that contain aggregate functions are properly set up for aggregate processing, allowing the case operation to participate correctly in aggregate expression evaluation.

22.14.6 TCaseOperation.UpdateAggregate

Synopsis: Update aggregate calculations for case operation

Declaration: `procedure UpdateAggregate;  Override`

Visibility: public

Description: The `UpdateAggregate` method updates aggregate calculations for the case operation node during aggregate expression evaluation. It ensures that aggregate functions within the selected case branch are properly updated with current data values.

The update process involves evaluating the condition to determine the active case branch, then updating any aggregate functions contained within that branch, allowing proper aggregate calculation in conditional expressions.

22.14.7 TCaseOperation.HasAggregate

Synopsis: Check if case operation contains aggregates

Declaration: `function HasAggregate : Boolean;  Override`

Visibility: public

Description: The HasAggregate (704) method determines whether the case operation contains any aggregate functions within its condition or case branches. It returns true if any part of the case operation uses aggregate functions.

22.14.8 TCaseOperation.NodeType

Synopsis: Get the result type for case operation

Declaration: `function NodeType : TResultType;  Override`

Visibility: public

Description: The `NodeType` method returns the data type that will be produced by the case operation. The result type is determined by analyzing all possible case branches to ensure they are compatible and determining the common result type.

The case operation ensures that all branches produce results of the same type, or automatically performs appropriate type conversions to achieve a common result type for the entire operation.

22.14.9 TCaseOperation.Create

Synopsis: Create case operation with arguments

Declaration: `constructor Create(Args: TExprArgumentArray)`

Visibility: public

Description: The `Create` constructor creates a new case operation node with the provided arguments array. The arguments typically include the condition expression and alternating case values and result expressions.

The case operation follows a pattern commonly found in SQL:

```
CASE condition
  WHEN value1 THEN result1
  WHEN value2 THEN result2
ELSE
  default_result
END.
```

The arguments array contains these components in the appropriate order for evaluation.

22.14.10 TCaseOperation.destroy

Synopsis: Destroy case operation and free resources

Declaration: `destructor destroy;  Override`

Visibility: public

Description: `Destroy` cleans up the case operation and frees all associated resources. It properly deallocates all child expression nodes including the condition and all case branches.

22.14.11 TCaseOperation.AsString

Synopsis: Get string representation of case operation

Declaration: `function AsString : AnsiString;  Override`

Visibility: public

Description: The AsString (705) method returns a human-readable string representation of the case operation, showing the condition and all case branches in a format that can be used for debugging or display purposes.

The string format typically follows the pattern:

```
CASE condition
  WHEN value1 THEN result1
  WHEN value2 THEN result2
ELSE
  default_result
END.
```

making the case logic clearly visible for analysis and debugging.

22.14.12 TCaseOperation.Condition

Synopsis: The condition expression of the case operation

Declaration: `Property Condition : TFPExprNode`

Visibility: public

Access: Read

Description: The Condition (705) property provides access to the condition expression that determines which case branch will be executed. The condition is evaluated and its result is compared against the WHEN values to select the appropriate branch.

The condition expression can be any valid expression that produces a value compatible with the WHEN clause values, allowing flexible case-based conditional logic.

22.15 TCurrencyToDateTimeNode

22.15.1 Description

The `TCurrencyToDateTimeNode` conversion node class handles the conversion of currency values to date/time values in expressions. It treats the currency value as a numeric representation that can be converted to datetime format.

This class is used internally by the TFPExpressionParser (728) class to construct an AST (Abstract Syntax Tree). There is normally no need for you to use or instantiate this class directly.

See also: TFPExprNode (750)

22.15.2 Method overview

Page	Method	Description
706	Check	Validate the currency to datetime conversion
707	GetNodeValue	Convert currency value to datetime
706	NodeType	Get the result type for currency to datetime conversion

22.15.3 TCurrencyToDateTimeNode.Check

Synopsis: Validate the currency to datetime conversion

Declaration: `procedure Check;  Override`

Visibility: public

Description: The `Check` method validates that the currency value can be safely converted to a datetime value, checking for valid datetime ranges and ensuring the conversion will produce a valid date/time result.

22.15.4 TCurrencyToDateTimeNode.NodeType

Synopsis: Get the result type for currency to datetime conversion

Declaration: `function NodeType : TResultType;  Override`

Visibility: public

Description: The `NodeType` method returns rtDateTime (691) to indicate that this conversion produces datetime results, enabling proper datetime handling in subsequent operations.

22.15.5 TCurrencyToDateTimeNode.GetNodeValue

Synopsis: Convert currency value to datetime

Declaration: `procedure GetNodeValue(var Result: TFPExpressionResult);  Override`

Visibility: public

Description: The `GetNodeValue` method converts a currency value to a datetime value by treating the currency amount as a numeric datetime representation, applying the standard Pascal datetime encoding.

22.16 TCurrencyToFloatNode

22.16.1 Description

The `TCurrencyToFloatNod` conversion node class handles the conversion of currency values to floating-point values in expressions. It preserves the numeric value while changing the data type to enable mathematical operations that require floating-point operands.

This class is used internally by the TFPExpressionParser (728) class to construct an AST (Abstract Syntax Tree). There is normally no need for you to use or instantiate this class directly.

See also: TFPExprNode (750)

22.16.2 Method overview

Page	Method	Description
707	Check	Validate the currency to float conversion
708	GetNodeValue	Convert currency value to float
707	NodeType	Get the result type for currency to float conversion

22.16.3 TCurrencyToFloatNode.Check

Synopsis: Validate the currency to float conversion

Declaration: `procedure Check;  Override`

Visibility: public

Description: The `Check` method validates that the currency value can be safely converted to a floating-point value, ensuring that precision is maintained and no overflow conditions occur during the conversion process.

22.16.4 TCurrencyToFloatNode.NodeType

Synopsis: Get the result type for currency to float conversion

Declaration: `function NodeType : TResultType;  Override`

Visibility: public

Description: The `NodeType` method returns rtFloat (691) to indicate that this conversion produces floating-point results, enabling proper type handling for subsequent mathematical operations.

22.16.5 TCurrencyToFloatNode.GetNodeValue

Synopsis: Convert currency value to float

Declaration: `procedure GetNodeValue(var Result: TFPExpressionResult); Override`

Visibility: public

Description: The `GetNodeValue` method performs the actual conversion of a currency value to a floating-point value, maintaining the numeric precision while changing the data type representation for mathematical compatibility.

22.17 TExprBuiltInManager

22.17.1 Description

The `TExprBuiltInManager` class serves as a manager for built-in functions and predefined variables that can be used within expressions. It provides a centralized registry where built-in mathematical functions, string functions, date/time functions, and system variables are registered and made available to the expression parser.

The manager handles the registration, lookup, and instantiation of built-in functions when they are encountered during expression parsing. It also maintains predefined variables such as system constants that can be referenced in expressions.

The `TExprBuiltInManager` class allows for extensibility by enabling the registration of custom built-in functions and variables that should be available across all expression evaluations.

See also: TFPExpressionParser (728)

22.17.2 Method overview

Page	Method	Description
710	AddBooleanVariable	Add boolean variable to built-in collection
711	AddCurrencyVariable	Add currency variable to built-in collection
711	AddDateTimeVariable	Add datetime variable to built-in collection
710	AddFloatVariable	Add float variable to built-in collection
711	AddFunction	Add function to built-in collection
710	AddIntegerVariable	Add integer variable to built-in collection
711	AddStringVariable	Add string variable to built-in collection
710	AddVariable	Add variable to built-in collection
709	Create	Create built-in manager instance
712	Delete	
709	Destroy	Destroy built-in manager instance
709	FindIdentifier	Find built-in identifier by name
709	IdentifierByName	Get identifier by name with exception on not found
709	IndexOfIdentifier	Find index of built-in identifier
712	Remove	

22.17.3 Property overview

Page	Properties	Access	Description
712	IdentifierCount	r	Number of built-in identifiers
712	Identifiers	r	Array access to built-in identifiers

22.17.4 TExprBuiltInManager.Create

Synopsis: Create built-in manager instance

Declaration:
```
constructor Create(AOwner: TComponent);  Override
```

Visibility: public

Description: The TExprBuiltInManager (708) constructor creates a new built-in manager instance that provides standard built-in functions and variables for expression parsing. The manager registers common mathematical, string, date/time, and conversion functions.

22.17.5 TExprBuiltInManager.Destroy

Synopsis: Destroy built-in manager instance

Declaration:
```
destructor Destroy;  Override
```

Visibility: public

Description: This destructor properly cleans up the built-in manager by freeing all registered built-in identifier definitions and any other allocated resources used by the manager.

22.17.6 TExprBuiltInManager.IndexOfIdentifier

Synopsis: Find index of built-in identifier

Declaration:
```
function IndexOfIdentifier(const AName: ShortString) : Integer
```

Visibility: public

Description: The IndexOfIdentifier (709) method searches for a built-in identifier with the specified name and returns its index position in the collection. If the identifier is not found, it returns -1. The search is case-sensitive.

22.17.7 TExprBuiltInManager.FindIdentifier

Synopsis: Find built-in identifier by name

Declaration:
```
function FindIdentifier(const AName: ShortString)
                            : TFPBuiltInExprIdentifierDef
```

Visibility: public

Description: The FindIdentifier (709) method searches for a built-in identifier with the specified name and returns the TFPBuiltInExprIdentifierDef (722) object if found, or nil if not found. The search is case-sensitive and covers all registered built-in functions and variables.

22.17.8 TExprBuiltInManager.IdentifierByName

Synopsis: Get identifier by name with exception on not found

Declaration:
```
function IdentifierByName(const AName: ShortString)
                                : TFPBuiltInExprIdentifierDef
```

Visibility: public

Description: The IdentifierByName (709) method retrieves a built-in identifier by name from the manager. Unlike FindIdentifier (709), this method raises an exception if the identifier is not found, making it suitable when the identifier is expected to exist.

22.17.9 TExprBuiltInManager.AddVariable

Synopsis: Add variable to built-in collection

Declaration:
```
function AddVariable(const ACategory: TBuiltInCategory;
                     const AName: ShortString; AResultType: TResultType;
                     const AValue: AnsiString)
                      : TFPBuiltInExprIdentifierDef
```

Visibility: public

Description: The AddVariable (710) method creates and adds a new built-in variable to the manager with the specified category, name, and value. The variable becomes available for use in expressions parsed by this manager.

22.17.10 TExprBuiltInManager.AddBooleanVariable

Synopsis: Add boolean variable to built-in collection

Declaration:
```
function AddBooleanVariable(const ACategory: TBuiltInCategory;
                            const AName: ShortString; AValue: Boolean)
                             : TFPBuiltInExprIdentifierDef
```

Visibility: public

Description: The AddBooleanVariable (710) method creates and adds a new boolean built-in variable to the manager with the specified category, name, and boolean value. The variable can then be referenced in expressions to retrieve the boolean value.

22.17.11 TExprBuiltInManager.AddIntegerVariable

Synopsis: Add integer variable to built-in collection

Declaration:
```
function AddIntegerVariable(const ACategory: TBuiltInCategory;
                            const AName: ShortString; AValue: Integer)
                             : TFPBuiltInExprIdentifierDef
```

Visibility: public

Description: The AddIntegerVariable (710) method creates and adds a new integer built-in variable to the manager with the specified category, name, and integer value. The variable can then be referenced in expressions to retrieve the numeric value.

22.17.12 TExprBuiltInManager.AddFloatVariable

Synopsis: Add float variable to built-in collection

Declaration:
```
function AddFloatVariable(const ACategory: TBuiltInCategory;
                          const AName: ShortString; AValue: TExprFloat)
                           : TFPBuiltInExprIdentifierDef
```

Visibility: public

Description: The AddFloatVariable (710) method creates and adds a new floating-point built-in variable to the manager with the specified category, name, and float value. The variable can then be referenced in expressions to retrieve the floating-point value.

22.17.13 TExprBuiltInManager.AddCurrencyVariable

Synopsis: Add currency variable to built-in collection

Declaration:
```
function AddCurrencyVariable(const ACategory: TBuiltInCategory;
                             const AName: ShortString; AValue: Currency)
                              : TFPBuiltInExprIdentifierDef
```

Visibility: public

Description: The AddCurrencyVariable (711) method creates and adds a new currency built-in variable to the manager with the specified category, name, and currency value. The variable can then be referenced in expressions to retrieve the monetary value.

22.17.14 TExprBuiltInManager.AddStringVariable

Synopsis: Add string variable to built-in collection

Declaration:
```
function AddStringVariable(const ACategory: TBuiltInCategory;
                           const AName: ShortString;
                           const AValue: AnsiString)
                            : TFPBuiltInExprIdentifierDef
```

Visibility: public

Description: The AddStringVariable (711) method creates and adds a new string built-in variable to the manager with the specified category, name, and string value. The variable can then be referenced in expressions to retrieve the text value.

22.17.15 TExprBuiltInManager.AddDateTimeVariable

Synopsis: Add datetime variable to built-in collection

Declaration:
```
function AddDateTimeVariable(const ACategory: TBuiltInCategory;
                             const AName: ShortString; AValue: TDateTime)
                              : TFPBuiltInExprIdentifierDef
```

Visibility: public

Description: The AddDateTimeVariable (711) method adds a datetime variable with the specified name and initial value to the built-in collection. The variable will be available in expressions under the provided name and will initially hold the specified datetime value.

Built-in datetime variables can be used to provide predefined date/time values like current date, system timestamps, or reference dates that expressions can access and use in date/time calculations.

22.17.16 TExprBuiltInManager.AddFunction

Synopsis: Add function to built-in collection

Declaration:
```
function AddFunction(const ACategory: TBuiltInCategory;
                     const AName: ShortString;
                     const AResultType: AnsiChar;
                     const AParamTypes: AnsiString;
                     ACallBack: TFPExprFunctionCallBack)
                      : TFPBuiltInExprIdentifierDef
function AddFunction(const ACategory: TBuiltInCategory;
```

```
                      const AName: ShortString;
                      const AResultType: AnsiChar;
                      const AParamTypes: AnsiString;
                      ACallBack: TFPExprFunctionEvent)
                       : TFPBuiltInExprIdentifierDef
function AddFunction(const ACategory: TBuiltInCategory;
                      const AName: ShortString;
                      const AResultType: AnsiChar;
                      const AParamTypes: AnsiString;
                      ANodeClass: TFPExprFunctionClass)
                       : TFPBuiltInExprIdentifierDef
```

Visibility: public

Description: The `AddFunction` method adds a custom function to the built-in collection, making it available for use in expressions. The function must provide appropriate callback or event handling mechanisms for evaluation.

Built-in functions extend the expression language with domain-specific functionality, allowing applications to provide mathematical, string, date/time, or business-logic functions that can be called from within expressions.

22.17.17 TExprBuiltInManager.Delete

Declaration: `procedure Delete(AIndex: Integer)`

Visibility: public

22.17.18 TExprBuiltInManager.Remove

Declaration: `function Remove(const aIdentifier: AnsiString) : Integer`

Visibility: public

22.17.19 TExprBuiltInManager.IdentifierCount

Synopsis: Number of built-in identifiers

Declaration: `Property IdentifierCount : Integer`

Visibility: public

Access: Read

Description: The `IdentifierCount` property returns the total number of built-in identifiers (functions and variables) currently registered in the manager. It includes all predefined mathematical functions, string functions, date/time functions, and variables.

This count is useful for iterating through all available built-ins or for applications that need to report or analyze the available expression language features.

22.17.20 TExprBuiltInManager.Identifiers

Synopsis: Array access to built-in identifiers

Declaration: `Property Identifiers[AIndex: Integer]: TFPBuiltInExprIdentifierDef`

Visibility: public

Access: Read

Description: The `Identifiers` property provides indexed access to the built-in identifier definitions registered in the manager. Each element is a TFPExprIdentifierDef (739) object containing function or variable information.

The array allows applications to enumerate and examine all available built-in functions and variables, useful for creating help systems, function browsers, or validation tools.

22.18 TFloatToCurrencyNode

22.18.1 Description

The `TFloatToCurrencyNode` conversion node class handles the conversion of floating-point values to currency values in expressions.

This class is used internally by the TFPExpressionParser (728) class to construct an AST (Abstract Syntax Tree). There is normally no need for you to use or instantiate this class directly.

See also: TFPExprNode (750)

22.18.2 Method overview

Page	Method	Description
713	Check	Validate the float to currency conversion
714	GetNodeValue	Convert float value to currency
713	NodeType	Get the result type for float to currency conversion

22.18.3 TFloatToCurrencyNode.Check

Synopsis: Validate the float to currency conversion

Declaration: `procedure Check;  Override`

Visibility: public

Description: The `Check` method validates that the floating-point value can be safely converted to a currency value, checking for valid currency ranges and preventing overflow conditions that could result in incorrect currency amounts.

22.18.4 TFloatToCurrencyNode.NodeType

Synopsis: Get the result type for float to currency conversion

Declaration: `function NodeType : TResultType;  Override`

Visibility: public

Description: The `NodeType` method returns rtCurrency (691) to indicate that this conversion produces currency results, enabling proper currency handling in the expression parser.

22.18.5 TFloatToCurrencyNode.GetNodeValue

Synopsis: Convert float value to currency

Declaration: `procedure GetNodeValue(var Result: TFPExpressionResult);  Override`

Visibility: public

Description: The `GetNodeValue` method performs the actual conversion of a floating-point value to a currency value, applying appropriate rounding and precision rules to ensure accurate currency representation.

22.19 TFloatToDateTimeNode

22.19.1 Description

The `TFloatToDateTimeNode` conversion node class handles the conversion of floating-point values to date/time values in expressions. It supports the standard Pascal datetime representation where the integer part represents days and the fractional part represents time.

The class enables precise datetime calculations by allowing floating-point values to represent both date and time components in a single numeric value. This is particularly useful for calculations involving time intervals and fractional days.

This class is used internally by the TFPExpressionParser (728) class to construct an AST (Abstract Syntax Tree). There is normally no need for you to use or instantiate this class directly.

See also: TFPExprNode (750)

22.19.2 Method overview

Page	Method	Description
714	Check	Validate the float to datetime conversion
715	GetNodeValue	Convert float value to datetime
714	NodeType	Get the result type for float to datetime conversion

22.19.3 TFloatToDateTimeNode.Check

Synopsis: Validate the float to datetime conversion

Declaration: `procedure Check;  Override`

Visibility: public

Description: The `Check` method validates that the floating-point value can be safely converted to a datetime value. It checks for valid datetime ranges and ensures that the conversion will not result in invalid date/time values.

The validation helps prevent runtime errors that could occur when attempting to convert floating-point values that are outside the valid datetime range or represent invalid date combinations.

22.19.4 TFloatToDateTimeNode.NodeType

Synopsis: Get the result type for float to datetime conversion

Declaration: `function NodeType : TResultType;  Override`

Visibility: public

Description: The NodeType (714) method returns rtDateTime (691) to indicate that this conversion produces datetime results.

22.19.5 TFloatToDateTimeNode.GetNodeValue

Synopsis: Convert float value to datetime

Declaration: `procedure GetNodeValue(var Result: TFPExpressionResult);  Override`

Visibility: public

Description: The GetNodeValue (715) method converts a floating-point value to a datetime value using the Pascal datetime encoding. The integer part represents days since the base date, and the fractional part represents the time portion of the day.

22.20 TFPAddOperation

22.20.1 Description

The `TFPAddOperation` class implements the addition operation between two numeric operands. It extends the binary operation framework to provide addition-specific functionality for integers, floats, and currency values.

The operation handles type conversion automatically when operands have different numeric types, promoting the result to the more precise type as needed. For example, adding an integer to a float produces a float result.

The addition operation also supports string concatenation when both operands are strings, providing unified handling for both numeric addition and string concatenation under the same operator.

This class is used internally by the TFPExpressionParser (728) class to construct an AST (Abstract Syntax Tree). There is normally no need for you to use or instantiate this class directly.

See also: TFPExprNode (750)

22.20.2 Method overview

Page	Method	Description
715	AsString	Get string representation of addition operation

22.20.3 TFPAddOperation.AsString

Synopsis: Get string representation of addition operation

Declaration: `function AsString : AnsiString;  Override`

Visibility: public

Description: The `AsString` method returns a string representation of the addition operation, showing both operands connected by the "+" symbol for debugging and display purposes.

The string representation helps in understanding the structure of mathematical expressions and provides useful debugging information for complex arithmetic operations.

22.21 TFPBinaryAndOperation

22.21.1 Description

The `TFPBinaryAndOperation` class implements the logical AND operation between two boolean operands. It extends TFPBooleanOperation (720) to provide AND-specific functionality that evaluates to True only when both operands are True.

The operation uses short-circuit evaluation, meaning that if the left operand is False, the right operand is not evaluated since the result is already determined to be False. This optimization improves performance and can prevent evaluation of potentially expensive or problematic right operands.

This class is used internally by the TFPExpressionParser (728) class to construct an AST (Abstract Syntax Tree). There is normally no need for you to use or instantiate this class directly.

See also: TFPExprNode (750)

22.21.2 Method overview

Page	Method	Description
716	AsString	Get string representation of AND operation

22.21.3 TFPBinaryAndOperation.AsString

Synopsis: Get string representation of AND operation

Declaration: `function AsString : AnsiString;  Override`

Visibility: public

Description: The AsString (716) method returns a string representation of the AND operation, typically showing both operands connected by the "and" keyword for debugging and display purposes.

22.22 TFPBinaryOperation

22.22.1 Description

This `TFPBinaryOperation` is the base class for all binary operations in the expression parser. Binary operations take two operands (left and right) and perform some operation to produce a result. It extends TFPExprNode (750) to provide the foundation for all two-operand operations.

The class manages the operand nodes and provides common functionality such as aggregate handling, validation, and resource management that is shared by all binary operations.

Derived classes implement specific operations such as arithmetic operations (+, -, *, /), comparison operations (=, <, >), and logical operations (and, or, xor). Each operation defines its own evaluation logic while leveraging the common binary operation infrastructure.

This class is used internally by the TFPExpressionParser (728) class to construct an AST (Abstract Syntax Tree). There is normally no need for you to use or instantiate this class directly.

See also: TFPExprNode (750)

22.22.2 Method overview

Page	Method	Description
718	Check	Validate the binary operation for correctness
717	Create	Create binary operation with left and right operands
717	Destroy	Destroy binary operation and free resources
718	HasAggregate	Check if binary operation contains aggregates
717	InitAggregate	Initialize aggregate processing for binary operation
718	UpdateAggregate	Update aggregate calculations for binary operation

22.22.3 Property overview

Page	Properties	Access	Description
718	left	r	Left operand of the binary operation
719	Right	r	Right operand of the binary operation

22.22.4 TFPBinaryOperation.Create

Synopsis: Create binary operation with left and right operands

Declaration: `constructor Create(ALeft: TFPExprNode; ARight: TFPExprNode)`

Visibility: public

Description: The `TFPBinaryOperation` constructor creates a new binary operation with the specified left and right operands. It takes ownership of both operand nodes and manages their lifecycle as part of the binary operation.

The constructor properly initializes the binary operation structure and establishes the relationship between the operation and its operands, ensuring that the operands are properly managed during the operation's lifetime.

22.22.5 TFPBinaryOperation.Destroy

Synopsis: Destroy binary operation and free resources

Declaration: `destructor Destroy;  Override`

Visibility: public

Description: This destructor properly cleans up the binary operation by freeing both the left and right operand nodes and any other resources used by the operation.

The destructor ensures that all memory allocated for the operands is properly released, preventing memory leaks in complex expression trees.

22.22.6 TFPBinaryOperation.InitAggregate

Synopsis: Initialize aggregate processing for binary operation

Declaration: `procedure InitAggregate;  Override`

Visibility: public

Description: The `InitAggregate` method initializes aggregate processing for both the left and right operands of the binary operation. It delegates the aggregate initialization to both operands to ensure that any aggregate expressions within the operands are properly initialized.

The InitAggregate (717) method is called as part of the aggregate processing cycle when the expression contains aggregate functions that need special handling.

22.22.7 TFPBinaryOperation.UpdateAggregate

Synopsis: Update aggregate calculations for binary operation

Declaration: `procedure UpdateAggregate;  Override`

Visibility: public

Description: The `UpdateAggregate` method updates aggregate calculations for both the left and right operands of the binary operation. It delegates the aggregate updates to both operands to ensure that any aggregate expressions are properly maintained during the aggregation cycle.

The method is called repeatedly during aggregate processing to update the state of aggregate functions contained within the operands.

22.22.8 TFPBinaryOperation.HasAggregate

Synopsis: Check if binary operation contains aggregates

Declaration: `function HasAggregate : Boolean;  Override`

Visibility: public

Description: The `HasAggregate` method checks whether either the left or right operand contains aggregate expressions. It returns True if any of the operands has aggregate functionality, which indicates that special aggregate processing is required for this operation.

The method is used by the expression parser to identify operations that need aggregate handling and to ensure proper processing order for aggregate expressions.

22.22.9 TFPBinaryOperation.Check

Synopsis: Validate the binary operation for correctness

Declaration: `procedure Check;  Override`

Visibility: public

Description: The `Check` method validates the binary operation by checking both operands for correctness and ensuring that they are compatible for the specific operation. It calls the Check method on both operands to perform recursive validation.

The validation process ensures that the operation can be safely executed and that all type requirements and constraints are met before evaluation begins.

22.22.10 TFPBinaryOperation.left

Synopsis: Left operand of the binary operation

Declaration: `Property left : TFPExprNode`

Visibility: public

Access: Read

Description: The `left` property provides access to the left operand of the binary operation. The left operand is the first operand in the operation and is typically evaluated before the right operand in operations where evaluation order matters.

The property returns the TFPExprNode (750) that represents the left side of the binary operation, allowing access to its value and type information.

22.22.11 TFPBinaryOperation.Right

Synopsis: Right operand of the binary operation

Declaration: `Property Right : TFPExprNode`

Visibility: public

Access: Read

Description: The `Right` property provides access to the right operand of the binary operation. The right operand is the second operand in the operation and is typically evaluated after the left operand in operations where evaluation order matters.

The property returns the TFPExprNode (750) that represents the right side of the binary operation, allowing access to its value and type information.

22.23 TFPBinaryOrOperation

22.23.1 Description

The `TFPBinaryOrOperation` class implements the logical OR operation between two boolean operands. It extends TFPBooleanOperation (720) to provide OR-specific functionality that evaluates to True when at least one operand is True.

The operation uses short-circuit evaluation, meaning that if the left operand is True, the right operand is not evaluated since the result is already determined to be True. This optimization improves performance in expressions where the first condition is likely to be true.

The result is always a boolean value, following standard logical OR truth table rules where only False OR False produces False.

This class is used internally by the TFPExpressionParser (728) class to construct an AST (Abstract Syntax Tree). There is normally no need for you to use or instantiate this class directly.

See also: TFPExprNode (750)

22.23.2 Method overview

Page	Method	Description
719	AsString	Get string representation of OR operation

22.23.3 TFPBinaryOrOperation.AsString

Synopsis: Get string representation of OR operation

Declaration: `function AsString : AnsiString;  Override`

Visibility: public

Description: The `AsString` method returns a string representation of the OR operation, typically showing both operands connected by the "or" keyword for debugging and display purposes.

The string format aids in visualizing the logical structure of expressions and provides valuable information for debugging complex boolean logic.

22.24 TFPBinaryXOrOperation

22.24.1 Description

The `TFPBinaryXOrOperation` class implements the logical XOR (exclusive OR) operation between two boolean operands. It extends TFPBooleanOperation (720) to provide XOR-specific functionality that evaluates to True when exactly one operand is True.

The XOR operation returns True only when the operands have different boolean values (True XOR False or False XOR True), and returns False when both operands have the same value (True XOR True or False XOR False).

Unlike AND and OR operations, XOR requires evaluation of both operands since both values are needed to determine the result.

See also: TFPExprNode (750)

22.24.2 Method overview

Page	Method	Description
720	AsString	Get string representation of XOR operation

22.24.3 TFPBinaryXOrOperation.AsString

Synopsis: Get string representation of XOR operation

Declaration: `function AsString : AnsiString;  Override`

Visibility: public

Description: The `AsString` method returns a string representation of the XOR operation, typically showing both operands connected by the "xor" keyword for debugging and display purposes.

22.25 TFPBooleanOperation

22.25.1 Description

`TFPBooleanOperation` is the base class for binary operations that work with boolean operands and produce boolean results. It extends TFPBinaryOperation (716) to provide boolean-specific functionality and validation.

The class ensures that both operands are boolean expressions and provides the framework for logical operations such as AND, OR, and XOR. It handles type validation to ensure that only boolean values participate in logical operations.

Derived classes implement specific boolean operations while leveraging the common boolean operation infrastructure for validation and type management.

This class is used internally by the TFPExpressionParser (728) class to construct an AST (Abstract Syntax Tree). There is normally no need for you to use or instantiate this class directly.

See also: TFPExprNode (750)

22.25.2 Method overview

Page	Method	Description
721	Check	Validate the boolean operation for correctness
721	NodeType	Get the result type for boolean operations

22.25.3 TFPBooleanOperation.Check

Synopsis: Validate the boolean operation for correctness

Declaration: `procedure Check;  Override`

Visibility: public

Description: The `Check` method validates that both operands of the boolean operation are boolean expressions. It ensures type compatibility and prevents type mismatches that would result in runtime errors during evaluation.

The validation process checks that both the left and right operands produce boolean results, raising appropriate errors if non-boolean operands are found.

22.25.4 TFPBooleanOperation.NodeType

Synopsis: Get the result type for boolean operations

Declaration: `function NodeType : TResultType;  Override`

Visibility: public

Description: The `NodeType` method returns rtBoolean (691) to indicate that boolean operations always produce boolean results. This type information is used by the expression parser for type checking and result handling.

22.26 TFPBooleanResultOperation

22.26.1 Description

The `TFPBooleanResultOperation` class serves as the base class for binary operations that produce boolean results but may accept non-boolean operands. It extends TFPBinaryOperation (716) to provide the framework for comparison and relational operations.

Unlike TFPBooleanOperation (720) which requires boolean operands, this class allows various operand types while ensuring the result is always boolean. This makes it suitable for comparison operations like equality, less than, greater than, etc.

The class provides the infrastructure for operations that compare values of compatible types and return boolean results indicating the relationship between the operands.

This class is used internally by the TFPExpressionParser (728) class to construct an AST (Abstract Syntax Tree). There is normally no need for you to use or instantiate this class directly.

22.26.2 Method overview

Page	Method	Description
722	Check	Validate the boolean result operation
722	NodeType	Get the result type for boolean operations

22.26.3 TFPBooleanResultOperation.Check

Synopsis: Validate the boolean result operation

Declaration: `procedure Check;  Override`

Visibility: public

Description: The Check (722) method validates that the operands are compatible for the specific boolean result operation. While the result will be boolean, the operands may be of various types depending on the specific operation (e.g., numeric types for comparison operations).

The validation ensures that the operands are type-compatible for the comparison or relational operation being performed.

22.26.4 TFPBooleanResultOperation.NodeType

Synopsis: Get the result type for boolean operations

Declaration: `function NodeType : TResultType;  Override`

Visibility: public

Description: The NodeType (722) method returns rtBoolean (691) to indicate that operations derived from this class always produce boolean results, regardless of the input operand types.

22.27 TFPBuiltInExprIdentifierDef

22.27.1 Description

The `TFPBuiltInExprIdentifierDef` class extends TFPExprIdentifierDef (739) to provide additional functionality for built-in functions and variables. It includes category information and other metadata specific to built-in identifiers that come predefined with the expression parser.

22.27.2 Method overview

Page	Method	Description
723	Assign	Assign values from another built-in identifier definition

22.27.3 Property overview

Page	Properties	Access	Description
723	Category	rw	The category of the built-in identifier

22.27.4 TFPBuiltInExprIdentifierDef.Assign

Synopsis: Assign values from another built-in identifier definition

Declaration: `procedure Assign(Source: TPersistent);  Override`

Visibility: public

Description: The `Assign` method copies all properties and settings from another `TFPBuiltInExprIdentifierDef` object, as usual for a `TPersistent` descendant

22.27.5 TFPBuiltInExprIdentifierDef.Category

Synopsis: The category of the built-in identifier

Declaration: `Property Category : TBuiltInCategory`

Visibility: published

Access: Read,Write

Description: The `Category` property specifies which TBuiltInCategory (687) this built-in identifier belongs to. The category is used to group related functions together and can be used for selective inclusion or exclusion of function groups.

22.28 TFPConstExpression

22.28.1 Description

The `TFPConstExpression` class represents constant literal values in expressions, storing fixed values that do not change during expression evaluation. It provides the foundation for all literal constants including strings, integers, floats, booleans, and date/time values.

Constant expressions are the leaf nodes of expression trees that provide actual data values rather than computed results. They are essential building blocks that supply the raw data that other expression nodes operate on.

The class handles various data types through different constructor methods and provides type-safe storage and retrieval of constant values during expression evaluation.

This class is used internally by the TFPExpressionParser (728) class to construct an AST (Abstract Syntax Tree). There is normally no need for you to use or instantiate this class directly.

See also: TFPExprNode (750)

22.28.2 Method overview

Page	Method	Description
726	AsString	Get string representation of constant expression
725	Check	Validate the constant expression
725	CreateBoolean	Create constant expression with boolean value
725	CreateCurrency	Create constant expression with currency value
724	CreateDateTime	Create constant expression with datetime value
724	CreateFloat	Create constant expression with float value
724	CreateInteger	Create constant expression with integer value
724	CreateString	Create constant expression with string value
725	GetNodeValue	Get the constant value
725	NodeType	Get the result type for constant expressions

22.28.3 Property overview

Page	Properties	Access	Description
726	ConstValue	r	The constant value stored in the expression

22.28.4 TFPConstExpression.CreateString

Synopsis: Create constant expression with string value

Declaration: `constructor CreateString(const AValue: AnsiString)`

Visibility: public

Description: The `TFPConstExpression` constructor creates a constant expression that holds a string literal value. The string value is stored internally and will be returned whenever the expression is evaluated.

String constants are used for literal text values in expressions and can be used in string operations, comparisons, and function parameters that expect string arguments.

22.28.5 TFPConstExpression.CreateInteger

Synopsis: Create constant expression with integer value

Declaration: `constructor CreateInteger(AValue: Int64)`

Visibility: public

Description: The `TFPConstExpression` constructor creates a constant expression that holds an integer literal value. The integer value is stored internally and will be returned whenever the expression is evaluated.

Integer constants are used for whole number values in expressions and can be used in arithmetic operations, comparisons, and function parameters that expect integer arguments.

22.28.6 TFPConstExpression.CreateDateTime

Synopsis: Create constant expression with datetime value

Declaration: `constructor CreateDateTime(AValue: TDateTime)`

Visibility: public

Description: The `CreateDateTime` constructor creates a new constant expression node that holds a datetime value. The datetime constant can be used in expressions for date/time arithmetic, comparisons, and function parameters that expect datetime arguments.

22.28.7 TFPConstExpression.CreateFloat

Synopsis: Create constant expression with float value

Declaration: `constructor CreateFloat(AValue: TExprFloat)`

Visibility: public

Description: The `CreateFloat` constructor creates a new constant expression node that holds a floating-point value. The float constant can be used in expressions for mathematical calculations, comparisons, and function parameters that expect floating-point arguments.

22.28.8 TFPConstExpression.CreateBoolean

Synopsis: Create constant expression with boolean value

Declaration: `constructor CreateBoolean(AValue: Boolean)`

Visibility: public

Description: The `TFPConstExpression` constructor creates a new constant expression node that holds a boolean value (true or false).

22.28.9 TFPConstExpression.CreateCurrency

Synopsis: Create constant expression with currency value

Declaration: `constructor CreateCurrency(AValue: Currency)`

Visibility: public

Description: The `TFPConstExpression` constructor creates a new constant expression node that holds a currency value with fixed-point precision.

22.28.10 TFPConstExpression.Check

Synopsis: Validate the constant expression

Declaration: `procedure Check;  Override`

Visibility: public

Description: The `Check` method validates the constant expression to ensure that the stored constant value is valid and consistent with its declared type. For constant expressions, validation is typically straightforward since the value is fixed.

22.28.11 TFPConstExpression.NodeType

Synopsis: Get the result type for constant expressions

Declaration: `function NodeType : TResultType;  Override`

Visibility: public

Description: The `NodeType` method returns the data type of the constant value stored in the expression. The result type corresponds directly to the type of the constant data, whether it's integer, float, string, boolean, datetime, or currency.

For constant expressions, the node type is fixed and determined when the constant is created.

22.28.12 TFPConstExpression.GetNodeValue

Synopsis: Get the constant value

Declaration: `procedure GetNodeValue(var Result: TFPExpressionResult);  Override`

Visibility: public

Description: The `GetNodeValue` method returns the constant value stored in the expression as a TFPExpressionResult (695). Since constants have fixed values, this method always returns the same result regardless of context.

22.28.13 TFPConstExpression.AsString

Synopsis: Get string representation of constant expression

Declaration: `function AsString : AnsiString;  Override`

Visibility: public

Description: The `AsString` method returns a string representation of the constant expression, showing the constant value in a readable format. The string representation varies based on the data type of the constant value.

22.28.14 TFPConstExpression.ConstValue

Synopsis: The constant value stored in the expression

Declaration: `Property ConstValue : TFPExpressionResult`

Visibility: public

Access: Read

Description: The `ConstValue` property provides access to the constant value held by the expression node. The value is stored as a TFPExpressionResult (695) record containing both the value and its type information.

22.29 TFPConvertNode

22.29.1 Description

`TFPConvertNode` is the base class for all type conversion operations in the expression parser. It provides the fundamental infrastructure for converting values between different data types during expression evaluation.

The class extends TFPExprNode (750) to provide conversion-specific functionality that enables automatic type coercion and explicit type casting operations within expressions.

Derived classes implement specific conversion logic for different source and target type combinations, ensuring type safety and proper value transformation during expression evaluation.

This class is used internally by the TFPExpressionParser (728) class to construct an AST (Abstract Syntax Tree). There is normally no need for you to use or instantiate this class directly.

See also: TFPExprNode (750)

22.29.2 Method overview

Page	Method	Description
726	AsString	Get string representation of conversion operation

22.29.3 TFPConvertNode.AsString

Synopsis: Get string representation of conversion operation

Declaration: `function AsString : AnsiString;  Override`

Visibility: default

Description: The `AsString` method returns a string representation of the conversion operation, typically showing the source expression and the target type for debugging and display purposes.

The string representation helps in understanding the structure of complex expressions that include multiple type conversions and provides useful information for debugging expression parsing issues.

22.30 TFPDivideOperation

22.30.1 Description

The `TFPDivideOperation` class implements division operations between two numeric operands, handling division by zero checks and type conversion for integers, floats, and currency values.

This class is used internally by the TFPExpressionParser (728) class to construct an AST (Abstract Syntax Tree). There is normally no need for you to use or instantiate this class directly.

See also: TFPExprNode (750)

22.30.2 Method overview

Page	Method	Description
727	AsString	Get string representation of division operation
727	Check	Validate the division operation for correctness
728	GetNodeValue	Calculate the result of division operation
727	NodeType	Get the result type for division operations

22.30.3 TFPDivideOperation.Check

Synopsis: Validate the division operation for correctness

Declaration: `procedure Check;  Override`

Visibility: public

Description: The `Check` method validates that both operands are numeric types and checks for division by zero conditions to prevent runtime errors.

22.30.4 TFPDivideOperation.AsString

Synopsis: Get string representation of division operation

Declaration: `function AsString : AnsiString;  Override`

Visibility: public

Description: Returns string representation showing operands connected by "/" symbol for debugging purposes.

22.30.5 TFPDivideOperation.NodeType

Synopsis: Get the result type for division operations

Declaration: `function NodeType : TResultType;  Override`

Visibility: public

Description: Returns the appropriate result type for division operations, typically promoting to floating-point for precise division results.

22.30.6 TFPDivideOperation.GetNodeValue

Synopsis: Calculate the result of division operation

Declaration: `procedure GetNodeValue(var Result: TFPExpressionResult);  Override`

Visibility: public

Description: Performs the division calculation after checking for division by zero, returning the quotient with appropriate type handling.

22.31 TFPEqualOperation

22.31.1 Description

The `TFPEqualOperation` class implements the equality comparison operation (=) between two operands of compatible types. It extends TFPBooleanResultOperation (721) to provide equality-specific comparison logic.

The operation compares two values and returns True if they are equal, False otherwise. It handles different data types appropriately, using numeric equality for numbers, string comparison for strings, and boolean comparison for boolean values.

Type conversion is performed when necessary to compare values of different but compatible types (e.g., comparing an integer with a float).

This class is used internally by the TFPExpressionParser (728) class to construct an AST (Abstract Syntax Tree). There is normally no need for you to use or instantiate this class directly.

See also: TFPExprNode (750)

22.31.2 Method overview

Page	Method	Description
728	AsString	Get string representation of equality operation

22.31.3 TFPEqualOperation.AsString

Synopsis: Get string representation of equality operation

Declaration: `function AsString : AnsiString;  Override`

Visibility: public

Description: The AsString (728) method returns a string representation of the equality operation, showing both operands connected by the "=" symbol for debugging and display purposes.

22.32 TFPExpressionParser

22.32.1 Description

This is the main expression parser class that provides comprehensive functionality for parsing and evaluating mathematical expressions.

The parser handles the complete expression evaluation pipeline from tokenization through parsing to evaluation, supporting complex expressions with proper operator precedence, parentheses, function calls, and variable references.

Key features include built-in mathematical functions, string operations, date/time handling, boolean logic, comparison operations, and extensibility through custom functions and variables.

See also: TExprBuiltInManager (708)

22.32.2 Method overview

Page	Method	Description
730	Clear	Clear the current expression
729	Create	Create expression parser instance
729	Destroy	Destroy expression parser and free resources
731	Evaluate	Evaluate the current expression
730	EvaluateExpression	Evaluate the expression and return result
730	ExtractIdentifierNames	
730	ExtractNode	Extract expression node from result
731	HasAggregate	Check if expression contains aggregates
730	IdentifierByName	Get identifier definition by name
731	InitAggregate	Initialize aggregate calculation
731	ResultType	Get the result type of the expression
732	UpdateAggregate	Update aggregate calculation with value

22.32.3 Property overview

Page	Properties	Access	Description
733	AsBoolean	r	Expression result as boolean value
732	AsCurrency	r	Expression result as currency value
733	AsDateTime	r	Expression result as date/time value
732	AsFloat	r	Expression result as floating point value
732	AsInteger	r	Expression result as integer value
733	AsString	r	Expression result as string value
734	BuiltIns	rw	Manager for built-in functions and constants
733	Expression	rw	Expression string to be evaluated
734	Identifiers	rw	Collection of identifiers and their values

22.32.4 TFPExpressionParser.Create

Synopsis: Create expression parser instance

Declaration: `constructor Create(AOwner: TComponent);  Override`

Visibility: public

Description: The `Create` constructor creates a new expression parser instance that can parse and evaluate mathematical expressions. The parser is initialized with empty identifier collections and is ready to accept expression strings for parsing.

22.32.5 TFPExpressionParser.Destroy

Synopsis: Destroy expression parser and free resources

Declaration: `destructor Destroy;  Override`

Visibility: public

Description: This destructor properly cleans up the expression parser by freeing all allocated resources, including the parsed expression tree, identifier definitions, and any other internal structures used by the parser.

22.32.6 TFPExpressionParser.IdentifierByName

Synopsis: Get identifier definition by name

Declaration:
```
function IdentifierByName(const AName: ShortString)
                                  : TFPExprIdentifierDef;  Virtual
```

Visibility: public

Description: The `IdentifierByName` method searches for an identifier definition with the specified name in the parser's identifier collection. It returns the TFPExprIdentifierDef (739) object if found, or raises an exception if not found.

22.32.7 TFPExpressionParser.Clear

Synopsis: Clear the current expression

Declaration:
```
procedure Clear
```

Visibility: public

Description: The `Clear` method clears the currently parsed expression from the parser, freeing any expression tree that was previously built. After calling this method, the parser is ready to parse a new expression.

22.32.8 TFPExpressionParser.EvaluateExpression

Synopsis: Evaluate the expression and return result

Declaration:
```
procedure EvaluateExpression(out Result: TFPExpressionResult)
```

Visibility: public

Description: The `EvaluateExpression` method evaluates the currently parsed expression using the provided identifier values and returns the computed result. It processes the entire expression tree to produce the final result value.

22.32.9 TFPExpressionParser.ExtractIdentifierNames

Declaration:
```
procedure ExtractIdentifierNames(const aExpression: AnsiString;
                                 aList: TStringList);  Overload
procedure ExtractIdentifierNames(const aExpression: AnsiString;
                                 aCallback: TIdentifierEvent);  Overload
```

Visibility: public

22.32.10 TFPExpressionParser.ExtractNode

Synopsis: Extract expression node from result

Declaration:
```
function ExtractNode(var N: TFPExprNode) : Boolean
```

Visibility: public

Description: The `ExtractNode` method extracts and returns the expression tree node from an expression result structure. It provides access to the internal node representation for advanced manipulation or analysis of the parsed expression.

This allows you to evaluate the same expression multiple times without needing to re-parse the expresion: by changing e.g. the value of a variable and calling the TFPExpressionResult.NodeValue (695) method on the extracted node, you can get the updated result.

22.32.11 TFPExpressionParser.Evaluate

Synopsis: Evaluate the current expression

Declaration: `function Evaluate : TFPExpressionResult`

Visibility: public

Description: The `Evaluate` method evaluates the currently parsed expression and returns the result as an TFPExpressionResult (695). It executes the complete expression tree and computes the final value.

The evaluation process includes resolving variables, calling functions, and performing all arithmetic and logical operations according to the expression structure and operator precedence rules.

22.32.12 TFPExpressionParser.ResultType

Synopsis: Get the result type of the expression

Declaration: `function ResultType : TResultType`

Visibility: public

Description: The `ResultType` method returns the data type that the currently parsed expression will produce when evaluated. The result type is determined by analyzing the expression tree and applying type promotion rules.

22.32.13 TFPExpressionParser.HasAggregate

Synopsis: Check if expression contains aggregates

Declaration: `function HasAggregate : Boolean`

Visibility: public

Description: The `HasAggregate` method determines whether the current expression contains any aggregate functions such as Sum, Count, Average, Min, or Max. Aggregate functions require special handling during evaluation.

This information is important for determining the evaluation strategy, as expressions with aggregates may need multiple evaluation passes or special initialization steps.

22.32.14 TFPExpressionParser.InitAggregate

Synopsis: Initialize aggregate calculation

Declaration: `procedure InitAggregate`

Visibility: public

Description: The `InitAggregate` method initializes all aggregate functions within the expression for aggregate processing. It prepares aggregate functions like Sum, Count, Average, Min, and Max for data processing by resetting their internal state.

The `InitAggregate` method must be called before processing data sets with expressions containing aggregate functions to ensure correct aggregate calculations.

22.32.15 TFPExpressionParser.UpdateAggregate

Synopsis: Update aggregate calculation with value

Declaration: `procedure UpdateAggregate`

Visibility: public

Description: The `UpdateAggregate` method updates all aggregate functions in the expression with new data values during aggregate processing. It processes each data row or value set through the expression's aggregate functions.

The `TFPExpressionParser` method is called repeatedly for each data item when processing datasets with aggregate expressions to accumulate the aggregate results.

22.32.16 TFPExpressionParser.AsFloat

Synopsis: Expression result as floating point value

Declaration: `Property AsFloat : TExprFloat`

Visibility: public

Access: Read

Description: The `AsFloat` property returns the expression evaluation result converted to a floating-point value. It automatically handles type conversion from the expression's native result type to float.

22.32.17 TFPExpressionParser.AsCurrency

Synopsis: Expression result as currency value

Declaration: `Property AsCurrency : Currency`

Visibility: public

Access: Read

Description: The `AsCurrency` property returns the expression evaluation result converted to a currency value with fixed-point precision. It handles automatic type conversion from compatible numeric types to currency.

22.32.18 TFPExpressionParser.AsInteger

Synopsis: Expression result as integer value

Declaration: `Property AsInteger : Int64`

Visibility: public

Access: Read

Description: The `AsInteger` property returns the expression evaluation result converted to an integer value. It automatically handles type conversion from the expression's native result type to integer, applying appropriate rounding or truncation.

22.32.19 TFPExpressionParser.AsString

Synopsis: Expression result as string value

Declaration: `Property AsString : AnsiString`

Visibility: public

Access: Read

Description: The AsString (733) property returns the expression evaluation result converted to a string representation. It automatically handles type conversion from any result type to a human-readable string format.

22.32.20 TFPExpressionParser.AsBoolean

Synopsis: Expression result as boolean value

Declaration: `Property AsBoolean : Boolean`

Visibility: public

Access: Read

Description: The `AsBoolean` property returns the expression evaluation result converted to a boolean value. It handles automatic type conversion from compatible types to boolean, following standard conversion rules.

22.32.21 TFPExpressionParser.AsDateTime

Synopsis: Expression result as date/time value

Declaration: `Property AsDateTime : TDateTime`

Visibility: public

Access: Read

Description: The `AsDateTime` property returns the expression evaluation result converted to a date/time value. It automatically handles type conversion from compatible numeric or string types to datetime format.

22.32.22 TFPExpressionParser.Expression

Synopsis: Expression string to be evaluated

Declaration: `Property Expression : Ansistring`

Visibility: published

Access: Read,Write

Description: The `Expression` property contains the expression text that will be parsed and evaluated by the parser. Setting this property triggers the parsing process, converting the text into an internal expression tree representation.

The expression string follows the syntax rules supported by the parser, including operators, functions, variables, and constants. Any syntax errors in the expression will be reported during the parsing phase.

22.32.23 TFPExpressionParser.Identifiers

Synopsis: Collection of identifiers and their values

Declaration: `Property Identifiers : TFPExprIdentifierDefs`

Visibility: published

Access: Read,Write

Description: The `Identifiers` property provides access to the collection of user-defined identifiers (variables and functions) that can be referenced within expressions. It contains TFPExprIdentifierDef (739) objects defining each identifier.

Applications can add custom variables and functions to this collection to extend the expression language with domain-specific functionality and data sources.

22.32.24 TFPExpressionParser.BuiltIns

Synopsis: Manager for built-in functions and constants

Declaration: `Property BuiltIns : TBuiltInCategories`

Visibility: published

Access: Read,Write

Description: The `BuiltIns` property references the TExprBuiltInManager (708) instance that p rovides access to built-in mathematical functions, string functions, date/time functions, and predefined constants.

The built-ins manager can be customized to add or modify the set of built-in functions available to expressions, allowing applications to tailor the function library to their specific needs.

22.33 TFPExpressionScanner

22.33.1 Description

The `TFPExpressionScanner` class performs lexical analysis on expression strings, breaking them into tokens that can be consumed by the expression parser. It handles the low-level details of character recognition, string parsing, and token classification.

The scanner recognizes all token types defined in TTokenType (691) including operators, literals, identifiers, keywords, and delimiters.

22.33.2 Method overview

Page	Method	Description
735	Create	Create a new expression scanner
735	GetToken	Get the next token from the source expression

22.33.3 Property overview

Page	Properties	Access	Description
736	CurrentChar	r	The current character being processed
736	Pos	r	Current position in the source string
736	Source	rw	The source expression string to scan
735	Token	r	The text of the current token
735	TokenType	r	The type of the current token

22.33.4 TFPExpressionScanner.Create

Synopsis: Create a new expression scanner

Declaration: `constructor Create`

Visibility: public

Description: The `Create` constructor creates a new expression scanner instance and initializes its internal state for tokenizing expression strings.

22.33.5 TFPExpressionScanner.GetToken

Synopsis: Get the next token from the source expression

Declaration: `function GetToken : TTokenType`

Visibility: public

Description: The `GetToken` method advances the scanner to the next token in the source expression and returns the token type. The token text is made available through the Token property.

The method handles all aspects of tokenization including whitespace skipping, literal parsing, operator recognition, and error detection for invalid character sequences.

22.33.6 TFPExpressionScanner.Token

Synopsis: The text of the current token

Declaration: `Property Token : AnsiString`

Visibility: public

Access: Read

Description: This read-only property provides access to the text content of the token that was most recently extracted by the scanner. For identifiers and literals, this contains the exact text from the expression; for operators and keywords, it contains the recognized symbol or word.

The property is updated automatically when GetToken (735) is called to advance to the next token.

22.33.7 TFPExpressionScanner.TokenType

Synopsis: The type of the current token

Declaration: `Property TokenType : TTokenType`

Visibility: public

Access: Read

Description: This read-only property returns the type classification of the current token using values from the TTokenType (691) enumeration. It indicates whether the token is an operator, identifier, literal, keyword, or delimiter.

The property is automatically updated when GetToken (735) advances to the next token.

22.33.8 TFPExpressionScanner.Source

Synopsis: The source expression string to scan

Declaration: `Property Source : AnsiString`

Visibility: public

Access: Read,Write

Description: The `Source` property gets or sets the expression string that will be tokenized by the scanner. When assigned, the scanner resets its position to the beginning and prepares for tokenizing the new expression.

Setting this property initializes the scanning process and makes the scanner ready to extract tokens from the expression string. The source string should contain a valid mathematical or logical expression using the supported syntax.

22.33.9 TFPExpressionScanner.Pos

Synopsis: Current position in the source string

Declaration: `Property Pos : Integer`

Visibility: public

Access: Read

Description: This read-only property returns the current character position within the source expression string during scanning. The position starts at 1 (first character) and advances as tokens are extracted.

The position value is useful for error reporting and debugging, as it indicates exactly where in the expression the scanner is currently located or where an error occurred during tokenization.

22.33.10 TFPExpressionScanner.CurrentChar

Synopsis: The current character being processed

Declaration: `Property CurrentChar : AnsiChar`

Visibility: public

Access: Read

Description: This read-only property returns the character at the current scanning position in the source expression string. It provides access to the character that will be processed next during tokenization.

The property is useful for debugging and for understanding what character the scanner is currently examining. When the scanner reaches the end of the source string, this property may return a null character or end-of-string marker.

22.34 TFPExprFunction

22.34.1 Description

The TFPExprFunction (737) class represents a function call node in the expression tree. It serves as the base class for all function implementations that can be called from within expressions, including mathematical functions, string functions, date/time functions, and user-defined functions.

The class manages function parameters, validates argument types and counts, and provides the framework for function evaluation. It handles the common aspects of function calls such as parameter management and type checking.

Derived classes implement specific functions by overriding the evaluation methods and providing function-specific logic. The class supports functions with varying numbers of parameters and different return types.

This class is used internally by the TFPExpressionParser (728) class to construct an AST (Abstract Syntax Tree). There is normally no need for you to use or instantiate this class directly.

See also: TFPExprNode (750)

22.34.2 Method overview

Page	Method	Description
739	AsString	Get string representation of function expression
737	Check	Validate the function expression
737	CreateFunction	Create function with identifier and arguments
738	Destroy	Destroy function expression and free resources
738	HasAggregate	Check if function contains aggregates
738	InitAggregate	Initialize aggregate processing for function
738	UpdateAggregate	Update aggregate calculations for function

22.34.3 Property overview

Page	Properties	Access	Description
739	ArgumentNodes	r	The argument nodes for the function
739	ArgumentParams	r	The argument parameters for the function

22.34.4 TFPExprFunction.Check

Synopsis: Validate the function expression

Declaration:
```
procedure Check;  Override
```

Visibility: public

Description: The `Check` method validates that the function expression is properly configured, with correct parameter count and types matching the function definition. It ensures that all arguments are valid expressions and that the function can be safely evaluated.

22.34.5 TFPExprFunction.CreateFunction

Synopsis: Create function with identifier and arguments

Declaration:
```
constructor CreateFunction(AID: TFPExprIdentifierDef;
                           const Args: TExprArgumentArray);  Virtual
```

Visibility: public

Description: The `CreateFunction` constructor creates a new function node with the specified identifier definition and argument expressions. It saves the function call structure and associates the function implementation with its parameters.

22.34.6 TFPExprFunction.Destroy

Synopsis: Destroy function expression and free resources

Declaration: `destructor Destroy;  Override`

Visibility: public

Description: This destructor properly cleans up the function expression by freeing all argument nodes and any other resources used by the function expression.

22.34.7 TFPExprFunction.InitAggregate

Synopsis: Initialize aggregate processing for function

Declaration: `procedure InitAggregate;  Override`

Visibility: public

Description: The `InitAggregate` method initializes aggregate processing for the function and all its arguments. It ensures that any aggregate operations within the function arguments are properly initialized for a new aggregate calculation cycle.

22.34.8 TFPExprFunction.UpdateAggregate

Synopsis: Update aggregate calculations for function

Declaration: `procedure UpdateAggregate;  Override`

Visibility: public

Description: The `UpdateAggregate` method updates aggregate calculations for the function and all its arguments. It ensures that any aggregate operations within the function arguments are properly updated during the aggregate processing cycle.

22.34.9 TFPExprFunction.HasAggregate

Synopsis: Check if function contains aggregates

Declaration: `function HasAggregate : Boolean;  Override`

Visibility: public

Description: The HasAggregate (738) method determines whether the function or any of its arguments contain aggregate operations. It returns True if aggregate processing is required for this function, indicating that special aggregate handling is needed.

22.34.10 TFPExprFunction.AsString

Synopsis: Get string representation of function expression

Declaration: `function AsString : AnsiString;  Override`

Visibility: public

Description: The `AsString` method returns a human-readable string representation of the function expression, including the function name and its arguments. The format typically shows the function call as it would appear in an expression.

22.34.11 TFPExprFunction.ArgumentNodes

Synopsis: The argument nodes for the function

Declaration: `Property ArgumentNodes : TExprArgumentArray`

Visibility: public

Access: Read

Description: The `ArgumentNodes` property provides access to the array of expression nodes that represent the function arguments. Each element is a TFPExprNode (750) containing the expression tree for one function parameter.

22.34.12 TFPExprFunction.ArgumentParams

Synopsis: The argument parameters for the function

Declaration: `Property ArgumentParams : TExprParameterArray`

Visibility: public

Access: Read

Description: The `ArgumentParams` property provides access to the array of evaluated parameter values for the function. Each element is a TFPExpressionResult (695) containing the evaluated value and type information for one function parameter.

22.35 TFPExprIdentifierDef

22.35.1 Description

The TFPExprIdentifierDef (739) class defines an identifier (variable or function) that can be used in expressions. It holds the name, type, and implementation details for identifiers, allowing the expression parser to resolve and evaluate them during expression processing.

22.35.2 Method overview

Page	Method	Description
740	ArgumentCount	Get the number of arguments for function identifier
740	Assign	Assign values from another identifier definition
740	EventBasedVariable	Check if identifier is an event-based variable

22.35.3 Property overview

Page	Properties	Access	Description
742	AsBoolean	rw	Get identifier value as boolean
741	AsCurrency	rw	Get identifier value as currency
742	AsDateTime	rw	Get identifier value as datetime
741	AsFloat	rw	Get identifier value as float
741	AsInteger	rw	Get identifier value as integer
741	AsString	rw	Get identifier value as string
743	IdentifierType	rw	The type of this identifier
743	Name	rw	The name of the identifier
744	NodeType	rw	The node type for this identifier
744	OnGetFunctionValue	rw	Event for function value evaluation
742	OnGetFunctionValueCallBack	rw	Callback property for function value evaluation
744	OnGetVariableValue	rw	Event for variable value evaluation
742	OnGetVariableValueCallBack	rw	Callback property for variable value evaluation
743	ParameterTypes	rw	The parameter types for function identifiers
744	ResultType	rw	The result type of the identifier
743	Value	rw	The value of the identifier
745	VariableArgumentCount	rw	

22.35.4 TFPExprIdentifierDef.ArgumentCount

Synopsis: Get the number of arguments for function identifier

Declaration: `function ArgumentCount : Integer`

Visibility: public

Description: The `ArgumentCount` method returns the number of arguments that a function identifier expects. For variable identifiers, it returns 0. For function identifiers, it returns the count of parameters required for the function call.

22.35.5 TFPExprIdentifierDef.Assign

Synopsis: Assign values from another identifier definition

Declaration: `procedure Assign(Source: TPersistent);  Override`

Visibility: public

Description: The Assign (740) method copies all properties and settings from another TFPExprIdentifierDef (739) object. It enables copying identifier definitions with all their associated configuration and callback information, as usual for a `TPersistent` descendant.

22.35.6 TFPExprIdentifierDef.EventBasedVariable

Synopsis: Check if identifier is an event-based variable

Declaration: `function EventBasedVariable : Boolean`

Visibility: public

Description: The `EventBasedVariable` method returns True if the identifier represents a variable that uses event-based evaluation rather than callback procedures. Event-based variables use method calls on objects for value resolution instead of standalone procedures.

22.35.7 TFPExprIdentifierDef.AsFloat

Synopsis: Get identifier value as float

Declaration: `Property AsFloat : TExprFloat`

Visibility: public

Access: Read,Write

Description: The `AsFloat` property returns the identifier's value converted to a floating-point number. If the identifier represents a variable, it retrieves the current value; if it represents a function, it calls the function and returns the result converted to float format.

22.35.8 TFPExprIdentifierDef.AsCurrency

Synopsis: Get identifier value as currency

Declaration: `Property AsCurrency : Currency`

Visibility: public

Access: Read,Write

Description: The `AsCurrency` property returns the identifier's value converted to a currency data type. If the identifier represents a variable, it retrieves the current value; if it represents a function, it calls the function and returns the result converted to currency format.

22.35.9 TFPExprIdentifierDef.AsInteger

Synopsis: Get identifier value as integer

Declaration: `Property AsInteger : Int64`

Visibility: public

Access: Read,Write

Description: The `AsInteger` property returns the identifier's value converted to an integer number. If the identifier represents a variable, it retrieves the current value; if it represents a function, it calls the function and returns the result converted to integer format.

22.35.10 TFPExprIdentifierDef.AsString

Synopsis: Get identifier value as string

Declaration: `Property AsString : AnsiString`

Visibility: public

Access: Read,Write

Description: The AsString (741) property returns the identifier's value converted to a string representation. If the identifier represents a variable, it retrieves the current value; if it represents a function, it calls the function and returns the result converted to string format.

22.35.11 TFPExprIdentifierDef.AsBoolean

Synopsis: Get identifier value as boolean

Declaration: `Property AsBoolean : Boolean`

Visibility: public

Access: Read,Write

Description: The `AsBoolean` property returns the identifier's value converted to a boolean value. If the identifier represents a variable, it retrieves the current value; if it represents a function, it calls the function and returns the result converted to boolean format.

22.35.12 TFPExprIdentifierDef.AsDateTime

Synopsis: Get identifier value as datetime

Declaration: `Property AsDateTime : TDateTime`

Visibility: public

Access: Read,Write

Description: The `AsDateTime` property returns the identifier's value converted to a date/time value. If the identifier represents a variable, it retrieves the current value; if it represents a function, it calls the function and returns the result converted to datetime format.

22.35.13 TFPExprIdentifierDef.OnGetFunctionValueCallBack

Synopsis: Callback property for function value evaluation

Declaration: `Property OnGetFunctionValueCallBack : TFPExprFunctionCallBack`

Visibility: public

Access: Read,Write

Description: The `OnGetFunctionValueCallBack` property holds a callback function that is called when the identifier represents a function and needs to be evaluated. The callback receives the function parameters and returns the computed result value.

22.35.14 TFPExprIdentifierDef.OnGetVariableValueCallBack

Synopsis: Callback property for variable value evaluation

Declaration: `Property OnGetVariableValueCallBack : TFPExprVariableCallBack`

Visibility: public

Access: Read,Write

Description: The `OnGetVariableValueCallBack` property holds a callback function that is called when the identifier represents a variable and its value needs to be retrieved. The callback returns the current value of the variable.

22.35.15 TFPExprIdentifierDef.IdentifierType

Synopsis: The type of this identifier

Declaration: `Property IdentifierType : TIdentifierType`

Visibility: published

Access: Read,Write

Description: The `IdentifierType` property specifies whether this identifier definition represents a variable or a function. The identifier type determines how the identifier will be processed during expression parsing and evaluation.

Variables are simple value holders, while functions require parameter processing and can perform complex calculations. The identifier type affects parsing behavior and determines which callback or evaluation method is used.

22.35.16 TFPExprIdentifierDef.Name

Synopsis: The name of the identifier

Declaration: `Property Name : ShortString`

Visibility: published

Access: Read,Write

Description: The `Name` property contains the name string that identifies this variable or function within expressions. The name is used during expression parsing to match identifier references to their definitions.

Identifier names are case-sensitive and must follow valid identifier naming conventions. This name serves as the key for looking up the identifier during expression evaluation.

22.35.17 TFPExprIdentifierDef.Value

Synopsis: The value of the identifier

Declaration: `Property Value : AnsiString`

Visibility: published

Access: Read,Write

Description: The `Value` property contains the current value of the identifier when it represents a variable. For variables, this property holds the actual data value that will be returned when the identifier is evaluated in expressions.

The value is stored as a string but is interpreted according to the identifier's result type. For function identifiers, this property is typically not used as functions compute their results dynamically.

22.35.18 TFPExprIdentifierDef.ParameterTypes

Synopsis: The parameter types for function identifiers

Declaration: `Property ParameterTypes : AnsiString`

Visibility: published

Access: Read,Write

Description: The `ParameterTypes` property specifies the expected parameter types for function identifiers. It contains an array or string describing the data types that the function expects to receive as arguments.

The parameter type information is used during expression parsing and validation to ensure that function calls provide arguments of the correct types, enabling compile-time type checking for function calls.

22.35.19 TFPExprIdentifierDef.ResultType

Synopsis: The result type of the identifier

Declaration: `Property ResultType : TResultType`

Visibility: published

Access: Read,Write

Description: The `ResultType` property specifies the data type that this identifier produces when evaluated. For variables, it indicates the type of the stored value. For functions, it indicates the type of value returned by the function.

22.35.20 TFPExprIdentifierDef.OnGetFunctionValue

Synopsis: Event for function value evaluation

Declaration: `Property OnGetFunctionValue : TFPExprFunctionEvent`

Visibility: published

Access: Read,Write

Description: The `OnGetFunctionValue` event is triggered when the identifier represents a function and needs to be evaluated with the provided parameters. The event handler receives the function arguments and should return the computed result.

22.35.21 TFPExprIdentifierDef.OnGetVariableValue

Synopsis: Event for variable value evaluation

Declaration: `Property OnGetVariableValue : TFPExprVariableEvent`

Visibility: published

Access: Read,Write

Description: The `OnGetVariableValue`event is triggered when the identifier represents a variable and its value needs to be retrieved during expression evaluation. The event handler should return the current value of the variable.

22.35.22 TFPExprIdentifierDef.NodeType

Synopsis: The node type for this identifier

Declaration: `Property NodeType : TFPExprFunctionClass`

Visibility: published

Access: Read,Write

Description: The `NodeType` property returns the result type that this identifier will produce when used in expressions. It corresponds to the ResultType (744) property and is used for expression tree type checking.

The node type information is essential for the expression parser to determine type compatibility and perform appropriate type conversions when this identifier is used in operations.

22.35.23 TFPExprIdentifierDef.VariableArgumentCount

Declaration: `Property VariableArgumentCount : Boolean`

Visibility: published

Access: Read,Write

22.36 TFPExprIdentifierDefs

22.36.1 Description

This collection class manages a set of TFPExprIdentifierDef (739) objects, providing methods for adding, removing, and searching identifier definitions. I t serves as a registry for all available variables and functions in an expression context.

22.36.2 Method overview

Page	Method	Description
746	AddBooleanVariable	Add a boolean variable to the collection
747	AddCurrencyVariable	Add a currency variable to the collection
748	AddDateTimeVariable	Add a datetime variable to the collection
747	AddFloatVariable	Add a float variable to the collection
748	AddFunction	Add a function to the collection
747	AddIntegerVariable	Add an integer variable to the collection
747	AddStringVariable	Add a string variable to the collection
746	AddVariable	Add a variable to the identifier collection
746	FindIdentifier	Find an identifier by name
746	IdentifierByName	Get an identifier by name
745	IndexOfIdentifier	Find the index of an identifier by name

22.36.3 Property overview

Page	Properties	Access	Description
748	Identifiers	rw	Access to identifier definitions by index

22.36.4 TFPExprIdentifierDefs.IndexOfIdentifier

Synopsis: Find the index of an identifier by name

Declaration: `function IndexOfIdentifier(const AName: ShortString) : Integer`

Visibility: public

Description: The `IndexOfIdentifier` method searches the collection for an identifier with the specified name and returns its index position. If no identifier with the given name is found, it returns -1.

The search is case-sensitive.

22.36.5 TFPExprIdentifierDefs.FindIdentifier

Synopsis: Find an identifier by name

Declaration:
```
function FindIdentifier(const AName: ShortString) : TFPExprIdentifierDef
```

Visibility: public

Description: The `FindIdentifier` method searches the collection for an identifier with the specified name and returns the TFPExprIdentifierDef (739) object if found, or nil if not found. The search is case-sensitive and provides a convenient way to locate identifier definitions.

22.36.6 TFPExprIdentifierDefs.IdentifierByName

Synopsis: Get an identifier by name

Declaration:
```
function IdentifierByName(const AName: ShortString)
                                  : TFPExprIdentifierDef
```

Visibility: public

Description: The `IdentifierByName` method retrieves an identifier by name from the collection. Unlike FindIdentifier (746), this method raises an exception if the identifier is not found, making it suitable for cases where the identifier is expected to exist.

22.36.7 TFPExprIdentifierDefs.AddVariable

Synopsis: Add a variable to the identifier collection

Declaration:
```
function AddVariable(const AName: ShortString;
                     AResultType: TResultType;
                     ACallback: TFPExprVariableCallBack)
                      : TFPExprIdentifierDef
function AddVariable(const AName: ShortString;
                     AResultType: TResultType;
                     ACallback: TFPExprVariableEvent)
                      : TFPExprIdentifierDef
function AddVariable(const AName: ShortString;
                     AResultType: TResultType; const AValue: AnsiString)
                      : TFPExprIdentifierDef
```

Visibility: public

Description: The `AddVariable` method creates and adds a new variable identifier to the collection with the specified name and value. The variable can then be referenced in expressions by its name to retrieve the stored value.

22.36.8 TFPExprIdentifierDefs.AddBooleanVariable

Synopsis: Add a boolean variable to the collection

Declaration:
```
function AddBooleanVariable(const AName: ShortString; AValue: Boolean)
                                  : TFPExprIdentifierDef
```

Visibility: public

Description: The `AddBooleanVariable` method creates and adds a new boolean variable identifier to the collection with the specified name and boolean value. The variable can then be referenced in expressions to retrieve the boolean value.

22.36.9 TFPExprIdentifierDefs.AddIntegerVariable

Synopsis: Add an integer variable to the collection

Declaration:
```
function AddIntegerVariable(const AName: ShortString; AValue: Integer)
                                  : TFPExprIdentifierDef
```

Visibility: public

Description: The `AddIntegerVariable` method creates and adds a new integer variable identifier to the collection with the specified name and integer value. The variable can then be referenced in expressions to retrieve the numeric value.

22.36.10 TFPExprIdentifierDefs.AddFloatVariable

Synopsis: Add a float variable to the collection

Declaration:
```
function AddFloatVariable(const AName: ShortString; AValue: TExprFloat)
                                : TFPExprIdentifierDef
```

Visibility: public

Description: The `AddFloatVariable` method creates and adds a new floating-point variable identifier to the collection with the specified name and float value. The variable can then be referenced in expressions to retrieve the floating-point value.

22.36.11 TFPExprIdentifierDefs.AddCurrencyVariable

Synopsis: Add a currency variable to the collection

Declaration:
```
function AddCurrencyVariable(const AName: ShortString; AValue: Currency)
                                  : TFPExprIdentifierDef
```

Visibility: public

Description: The `AddCurrencyVariable` method creates and adds a new currency variable identifier to the collection with the specified name and currency value. The variable can then be referenced in expressions to retrieve the monetary value.

22.36.12 TFPExprIdentifierDefs.AddStringVariable

Synopsis: Add a string variable to the collection

Declaration:
```
function AddStringVariable(const AName: ShortString;
                                const AValue: AnsiString)
                                 : TFPExprIdentifierDef
```

Visibility: public

Description: The `AddStringVariable` method creates and adds a new string variable identifier to the collection with the specified name and string value. The variable can then be referenced in expressions to retrieve the text value.

22.36.13 TFPExprIdentifierDefs.AddDateTimeVariable

Synopsis: Add a datetime variable to the collection

Declaration:
```
function AddDateTimeVariable(const AName: ShortString;
                             AValue: TDateTime) : TFPExprIdentifierDef
```

Visibility: public

Description: The `AddDateTimeVariable` method creates and adds a new date/time variable identifier to the collection with the specified name and datetime value. The variable can then be referenced in expressions to retrieve the date/time value.

22.36.14 TFPExprIdentifierDefs.AddFunction

Synopsis: Add a function to the collection

Declaration:
```
function AddFunction(const AName: ShortString;
                     const AResultType: AnsiChar;
                     const AParamTypes: AnsiString;
                     ACallBack: TFPExprFunctionCallBack)
                      : TFPExprIdentifierDef
function AddFunction(const AName: ShortString;
                     const AResultType: AnsiChar;
                     const AParamTypes: AnsiString;
                     ACallBack: TFPExprFunctionEvent)
                      : TFPExprIdentifierDef
function AddFunction(const AName: ShortString;
                     const AResultType: AnsiChar;
                     const AParamTypes: AnsiString;
                     ANodeClass: TFPExprFunctionClass)
                      : TFPExprIdentifierDef
```

Visibility: public

Description: The `AddFunction` method creates and adds a new function identifier to the collection with the specified name, parameter types, result type, and implementation details. The function becomes available for use in expressions parsed by this identifier collection.

22.36.15 TFPExprIdentifierDefs.Identifiers

Synopsis: Access to identifier definitions by index

Declaration:
```
Property Identifiers[AIndex: Integer]: TFPExprIdentifierDef; default
```

Visibility: public

Access: Read,Write

Description: This indexed property provides direct access to identifier definitions in the collection by their numeric index. It allows iteration through all identifiers and enables index-based retrieval of TFPExprIdentifierDef (739) objects.

22.37 TFPExprIdentifierNode

22.37.1 Description

The `TFPExprIdentifierNode` class represents an identifier reference in the expression tree. It holds a reference to the identifier definition and handles the resolution and evaluation of variables and function calls during expression evaluation.

This class is used internally by the TFPExpressionParser (728) class to construct an AST (Abstract Syntax Tree). There is normally no need for you to use or instantiate this class directly.

See also: TFPExprNode (750)

22.37.2 Method overview

Page	Method	Description
749	CreateIdentifier	Create identifier node with identifier definition
750	GetNodeValue	Get the value of the identifier
749	NodeType	Get the result type for identifier nodes

22.37.3 Property overview

Page	Properties	Access	Description
750	Identifier	r	The identifier definition for this node

22.37.4 TFPExprIdentifierNode.CreateIdentifier

Synopsis: Create identifier node with identifier definition

Declaration: `constructor CreateIdentifier(AID: TFPExprIdentifierDef)`

Visibility: public

Description: The `CreateIdentifier` constructor creates a new identifier node and associates it with the specified TFPExprIdentifierDef (739) object. The identifier definition provides the name, type, and evaluation mechanism for the identifier.

22.37.5 TFPExprIdentifierNode.NodeType

Synopsis: Get the result type for identifier nodes

Declaration: `function NodeType : TResultType;  Override`

Visibility: public

Description: The NodeType (749) method returns the data type that this identifier node will produce when evaluated. The result type is determined by the associated identifier definition and its configured result type.

The node type is used by the expression parser for type checking and ensuring type compatibility throughout the expression tree during parsing and evaluation.

22.37.6 TFPExprIdentifierNode.GetNodeValue

Synopsis: Get the value of the identifier

Declaration: `procedure GetNodeValue(var Result: TFPExpressionResult); Override`

Visibility: public

Description: The `GetNodeValue` method retrieves the current value of the identifier by calling the appropriate evaluation mechanism (callback, event handler, or direct value access) based on the identifier's configuration.

For variables, this returns the stored value. For functions, this executes the function with any provided parameters and returns the computed result.

22.37.7 TFPExprIdentifierNode.Identifier

Synopsis: The identifier definition for this node

Declaration: `Property Identifier : TFPExprIdentifierDef`

Visibility: public

Access: Read

Description: The `Identifier` property references the TFPExprIdentifierDef (739) object that defines this identifier's characteristics, including its name, type, value, and evaluation methods.

The identifier definition contains all the metadata needed to properly evaluate this identifier node during expression evaluation, including callbacks, event handlers, and type information.

22.38 TFPExprNode

22.38.1 Description

This is the abstract base class for all nodes in the expression tree. Every element of a parsed expression (literals, variables, operators, functions) is represented by a class derived from `TFPExprNode`.

The class defines the common interface that all expression nodes must implement, including evaluation, type checking, and aggregate processing. It provides the foundation for building complex expression trees through composition of different node types.

Derived classes implement specific functionality for different types of expression elements such as arithmetic operations, function calls, variable references, and literal values.

22.38.2 Method overview

Page	Method	Description
753	AsString	Get string representation of the expression node
751	Check	Validate the expression node for correctness
752	HasAggregate	Check if node tree contains any aggregate operations
751	InitAggregate	Initialize aggregate processing for the node
751	IsAggregate	Check if this node represents an aggregate operation
752	NodeType	Get the result type of this expression node
752	NodeValue	Get the evaluated value of this expression node
751	UpdateAggregate	Update aggregate calculations for the node

22.38.3 TFPExprNode.Check

Synopsis: Validate the expression node for correctness

Declaration: `procedure Check; Virtual; Abstract`

Visibility: public

Description: The `Check` abstract method must be implemented by derived classes to validate the expression node and its operands or parameters. It performs type checking, verifies that required operands are present, and ensures that the node can be safely evaluated.

The validation process typically includes checking operand types for compatibility, verifying function parameter counts and types, and ensuring that all required references (variables, functions) are available. If validation fails, appropriate exceptions must be raised.

The Check (751) method is called during the expression preparation phase, before evaluation begins, to catch errors early and provide meaningful error messages.

22.38.4 TFPExprNode.InitAggregate

Synopsis: Initialize aggregate processing for the node

Declaration: `procedure InitAggregate; Virtual`

Visibility: public

Description: The `InitAggregate` virtual method initializes aggregate processing state for nodes that participate in aggregate operations. It is called at the beginning of an aggregate processing cycle to reset counters, accumulators, and other aggregate state.

For most expression nodes, this method does nothing (default implementation). However, nodes representing aggregate functions (like SUM, COUNT, AVG) override this method to initialize their aggregation state.

22.38.5 TFPExprNode.UpdateAggregate

Synopsis: Update aggregate calculations for the node

Declaration: `procedure UpdateAggregate; Virtual`

Visibility: public

Description: The `UpdateAggregate` virtual method updates the aggregate calculation state for nodes that participate in aggregate operations. It is called for each data row or iteration during aggregate processing to incorporate new values into the ongoing calculation.

For most expression nodes, this method does nothing (default implementation). Nodes representing aggregate functions override this method to update their running totals, counts, or other aggregate state based on the current data values.

The `UpdateAggregate` method works in conjunction with InitAggregate (751) to implement the aggregate processing cycle.

22.38.6 TFPExprNode.IsAggregate

Synopsis: Check if this node represents an aggregate operation

Declaration: `class function IsAggregate : Boolean; Virtual`

Visibility: public

Description: The `IsAggregate` class method returns True if the node class represents an aggregate operation that requires special processing during expression evaluation. Most node types return False, indicating they are simple expression elements.

Aggregate nodes (like SUM, COUNT, AVG functions) return True to indicate they need to participate in the aggregate processing cycle with initialization and update phases.

The `IsAggregate` method allows the expression evaluator to identify which nodes need aggregate processing and to ensure proper handling of aggregate expressions.

22.38.7 TFPExprNode.HasAggregate

Synopsis: Check if node tree contains any aggregate operations

Declaration: `function HasAggregate : Boolean;  Virtual`

Visibility: public

Description: The `HasAggregate` method examines the expression subtree rooted at this node to determine if it contains any aggregate operations. It returns True if this node or any of its descendant nodes represents an aggregate operation.

The method recursively checks child nodes to detect aggregate functions anywhere in the expression tree. This information is used by the expression evaluator to determine if aggregate processing is required for the expression.

Unlike IsAggregate (751) which only checks the current node, this method performs a search of the entire subtree.

22.38.8 TFPExprNode.NodeType

Synopsis: Get the result type of this expression node

Declaration: `function NodeType : TResultType;  Virtual;  Abstract`

Visibility: public

Description: The `NodeType` abstract method must be implemented by derived classes to return the TResultType (691) that indicates what type of value this node will produce when evaluated.

The result type information is essential for type checking during expression validation and for determining appropriate type conversions when combining values from different nodes.

Common result types include boolean, integer, float, string, datetime, and currency, depending on the specific operation or value represented by the node.

22.38.9 TFPExprNode.NodeValue

Synopsis: Get the evaluated value of this expression node

Declaration: `function NodeValue : TFPExpressionResult`

Visibility: public

Description: The `NodeValue` method evaluates the expression node and returns the result as a TFPExpressionResult (695) record containing both the value and its type information.

The method performs the actual computation represented by this node, including evaluating any child nodes or operands as needed. For simple nodes like literals, it returns the stored value; for complex nodes like operations, it computes the result from its operands.

The returned result includes type information to ensure proper handling of the value by calling code.

22.38.10 TFPExprNode.AsString

Synopsis: Get string representation of the expression node

Declaration: `function AsString : AnsiString; Virtual; Abstract`

Visibility: public

Description: The `AsString` abstract method must be implemented by derived classes to return a string representation of the expression node and its structure. The string typically shows the operation or value in a human-readable format.

For operations, the string usually includes the operands and operator (e.g., "A + B"). For literals, it shows the value. For function calls, it shows the function name and parameters.

The AsString (753) method is primarily used for debugging, logging, and displaying expression structures to users.

22.39 TFPExprVariable

22.39.1 Description

The `TFPExprVariable` class represents a variable reference in an expression tree. Variables are identifiers that hold values and can be referenced by name within expressions to retrieve their current values.

This class is used internally by the TFPExpressionParser (728) class to construct an AST (Abstract Syntax Tree). There is normally no need for you to use or instantiate this class directly.

See also: TFPExprNode (750)

22.39.2 Method overview

Page	Method	Description
754	AsString	Get string representation of variable expression
753	Check	Validate the variable expression

22.39.3 TFPExprVariable.Check

Synopsis: Validate the variable expression

Declaration: `procedure Check; Override`

Visibility: default

Description: The `Check` method validates that the variable expression is properly configured and that the associated variable definition is valid and accessible for evaluation.

22.39.4 TFPExprVariable.AsString

Synopsis: Get string representation of variable expression

Declaration:
```
function AsString : AnsiString;  Override
```

Visibility: default

Description: The `AsString` method returns a human-readable string representation of the variable expression, typically showing the variable name as it appears in the original expression text.

22.40 TFPFunctionCallBack

22.40.1 Description

The `TFPFunctionCallBack` class implements function calls that are resolved using callback mechanisms rather than built-in function implementations. It extends the expression framework to allow dynamic function resolution through user-defined callback procedures.

The callback function mechanism enables applications to provide custom function implementations that can be called from within expressions. This allows for extending the expression language with application-specific functions without modifying the core expression parser.

When a callback function is called, the registered callback procedure is invoked with the function arguments and is expected to compute and return the function result.

See also: TFPExprNode (750)

22.40.2 Method overview

Page	Method	Description
754	CreateFunction	Create callback function with identifier and arguments
755	GetNodeValue	Get callback function result

22.40.3 Property overview

Page	Properties	Access	Description
755	CallBack	r	The callback procedure for function evaluation

22.40.4 TFPFunctionCallBack.CreateFunction

Synopsis: Create callback function with identifier and arguments

Declaration:
```
constructor CreateFunction(AID: TFPExprIdentifierDef;
                           const Args: TExprArgumentArray);  Override
```

Visibility: public

Description: The `CreateFunction` constructor creates a new callback function instance with the specified identifier definition and argument list. It establishes the connection between the function call and the callback mechanism that will handle the function execution.

The constructor stores the identifier information and argument expressions that will be used during function evaluation to invoke the appropriate callback procedure with the correct parameters.

22.40.5 TFPFunctionCallBack.GetNodeValue

Synopsis: Get callback function result

Declaration:
```
procedure GetNodeValue(var Result: TFPExpressionResult);  Override
```

Visibility: public

Description: The `GetNodeValue` method evaluates the callback function by calling the registered callback procedure with the function parameters and returns the computed result. It bridges the expression tree evaluation with external function implementations.

22.40.6 TFPFunctionCallBack.CallBack

Synopsis: The callback procedure for function evaluation

Declaration:
```
Property CallBack : TFPExprFunctionCallBack
```

Visibility: public

Access: Read

Description: The `CallBack` property holds the callback procedure that is invoked when the function needs to be evaluated. The callback receives the function parameters and returns the computed result, allowing external code to provide custom function implementations.

22.41 TFPFunctionEventHandler

22.41.1 Description

The `TFPFunctionEventHandler` class provides event-based function evaluation using object method calls instead of standalone callback procedures. It extends TFPExprFunction (737) to support method-based function implementations.

See also: TFPExprNode (750)

22.41.2 Method overview

Page	Method	Description
755	CreateFunction	Create event function with identifier and arguments
756	GetNodeValue	Get event function result

22.41.3 Property overview

Page	Properties	Access	Description
756	CallBack	r	The event handler for function evaluation

22.41.4 TFPFunctionEventHandler.CreateFunction

Synopsis: Create event function with identifier and arguments

Declaration:
```
constructor CreateFunction(AID: TFPExprIdentifierDef;
                           const Args: TExprArgumentArray);  Override
```

Visibility: public

Description: The `CreateFunction` constructor creates a new event-based function node with the specified identifier definition and argument expressions. It sets up the function call structure to use event handlers for function evaluation.

22.41.5 TFPFunctionEventHandler.GetNodeValue

Synopsis: Get event function result

Declaration: `procedure GetNodeValue(var Result: TFPExpressionResult); Override`

Visibility: public

Description: The GetNodeValue (756) method evaluates the event-based function by c alling the registered event handler method with the function parameters and returns the computed result. It provides method-based function evaluation.

22.41.6 TFPFunctionEventHandler.CallBack

Synopsis: The event handler for function evaluation

Declaration: `Property CallBack : TFPExprFunctionEvent`

Visibility: public

Access: Read

Description: The `CallBack` property holds the event handler method that is invoked when the function needs to be evaluated. The event handler receives the function parameters and returns the computed result, enabling object-based function implementations.

22.42 TFPGreaterThanEqualOperation

22.42.1 Description

The `TFPGreaterThanEqualOperation` class implements the greater-than-or-equal comparison operation (>=) between two operands of compatible types. It extends TFPOrderingOperation (764) to provide greater-than-or-equal-specific comparison logic.

The operation compares two values and returns True if the left operand is greater than or equal to the right operand, False otherwise. It handles different data types appropriately, using numeric comparison for numbers, lexicographic comparison for strings, and chronological comparison for date/time values.

This operation combines both equality and greater-than testing, returning True when either condition is satisfied.

See also: TFPExprNode (750)

22.42.2 Method overview

Page	Method	Description
757	AsString	Get string representation of greater-than-equal operation

22.42.3 TFPGreaterThanEqualOperation.AsString

Synopsis: Get string representation of greater-than-equal operation

Declaration: `function AsString : AnsiString;  Override`

Visibility: public

Description: The `AsString` method returns a string representation of the greater-than-or-equal operation, showing both operands connected by the ">=" symbol for debugging and display purposes.

The string format helps in visualizing comparison expressions and provides clear indication of the combined greater-than and equality test being performed.

22.43 TFPGreaterThanOperation

22.43.1 Description

The `TFPGreaterThanOperation` class implements the greater-than comparison operation (>) between two operands of compatible types. It extends TFPOrderingOperation (764) to provide greater-than-specific comparison logic.

The operation compares two values and returns True if the left operand is greater than the right operand, False otherwise. It handles different data types appropriately, using numeric comparison for numbers, lexicographic comparison for strings, and chronological comparison for date/time values.

Type conversion is performed when necessary to compare values of different but compatible types, ensuring proper ordering across numeric types.

This class is used internally by the TFPExpressionParser (728) class to construct an AST (Abstract Syntax Tree). There is normally no need for you to use or instantiate this class directly.

See also: TFPExprNode (750)

22.43.2 Method overview

Page	Method	Description
757	AsString	Get string representation of greater-than operation

22.43.3 TFPGreaterThanOperation.AsString

Synopsis: Get string representation of greater-than operation

Declaration: `function AsString : AnsiString;  Override`

Visibility: public

Description: The `AsString` method returns a string representation of the greater-than operation, showing both operands connected by the ">" symbol for debugging and display purposes.

The string format helps in visualizing comparison expressions and provides valuable information for debugging expression parsing and evaluation logic.

22.44 TFPLessThanEqualOperation

22.44.1 Description

The `TFPLessThanEqualOperation` class implements the less-than-or-equal comparison operation (<=) between two operands of compatible types. It extends TFPOrderingOperation (764) to provide less-than-or-equal-specific comparison logic.

The operation compares two values and returns True if the left operand is less than or equal to the right operand, False otherwise. It handles different data types appropriately, using numeric comparison for numbers, lexicographic comparison for strings, and chronological comparison for date/time values.

This operation combines both equality and less-than testing, returning True when either condition is satisfied.

See also: TFPExprNode (750)

22.44.2 Method overview

Page	Method	Description
758	AsString	Get string representation of less-than-equal operation

22.44.3 TFPLessThanEqualOperation.AsString

Synopsis: Get string representation of less-than-equal operation

Declaration: `function AsString : AnsiString;  Override`

Visibility: public

Description: The `AsString` method returns a string representation of the less-than-or-equal operation, showing both operands connected by the "<=" symbol for debugging and display purposes.

The string format helps in visualizing comparison expressions and provides clear indication of the combined less-than and equality test being performed.

22.45 TFPLessThanOperation

22.45.1 Description

The `TFPLessThanOperation` class implements the less-than comparison operation (<) between two operands of compatible types. It extends TFPOrderingOperation (764) to provide less-than-specific comparison logic.

The operation compares two values and returns True if the left operand is less than the right operand, False otherwise. It handles different data types appropriately, using numeric comparison for numbers, lexicographic comparison for strings, and chronological comparison for date/time values.

Type conversion is performed when necessary to compare values of different but compatible types, ensuring proper ordering across numeric types.

This class is used internally by the TFPExpressionParser (728) class to construct an AST (Abstract Syntax Tree). There is normally no need for you to use or instantiate this class directly.

See also: TFPExprNode (750)

22.45.2 Method overview

Page	Method	Description
759	AsString	Get string representation of less-than operation

22.45.3 TFPLessThanOperation.AsString

Synopsis: Get string representation of less-than operation

Declaration: `function AsString : AnsiString;  Override`

Visibility: public

Description: The `AsString` method returns a string representation of the less-than operation, showing both operands connected by the "<" symbol for debugging and display purposes.

The string format helps in visualizing comparison expressions and provides valuable information for debugging expression parsing and evaluation logic.

22.46 TFPModuloOperation

22.46.1 Description

The `TFPModuloOperation` class implements modulo operations that return the remainder after division between two integer operands.

This class is used internally by the TFPExpressionParser (728) class to construct an AST (Abstract Syntax Tree). There is normally no need for you to use or instantiate this class directly.

See also: TFPExprNode (750)

22.46.2 Method overview

Page	Method	Description
759	AsString	Get string representation of modulo operation
759	Check	Validate the modulo operation for correctness
760	GetNodeValue	Calculate the result of modulo operation
760	NodeType	Get the result type for modulo operations

22.46.3 TFPModuloOperation.Check

Synopsis: Validate the modulo operation for correctness

Declaration: `procedure Check;  Override`

Visibility: public

Description: Validates that both operands are integer types and checks for modulo by zero conditions.

22.46.4 TFPModuloOperation.AsString

Synopsis: Get string representation of modulo operation

Declaration: `function AsString : AnsiString;  Override`

Visibility: public

Description: Returns string representation showing operands connected by "mod" keyword for debugging purposes.

22.46.5 TFPModuloOperation.NodeType

Synopsis: Get the result type for modulo operations

Declaration: `function NodeType : TResultType;  Override`

Visibility: public

Description: Returns integer type as modulo operations work with and return integer values.

22.46.6 TFPModuloOperation.GetNodeValue

Synopsis: Calculate the result of modulo operation

Declaration: `procedure GetNodeValue(var Result: TFPExpressionResult);  Override`

Visibility: public

Description: Performs the modulo calculation returning the remainder after integer division.

22.47 TFPMultiplyOperation

22.47.1 Description

The `TFPMultiplyOperation` class implements multiplication operation between two numeric operands, handling type conversion and providing multiplication-specific functionality for integers, floats, and currency values.

This class is used internally by the TFPExpressionParser (728) class to construct an AST (Abstract Syntax Tree). There is normally no need for you to use or instantiate this class directly.

See also: TFPExprNode (750)

22.47.2 Method overview

Page	Method	Description
761	AsString	Get string representation of multiplication operation
760	check	Validate the multiplication operation for correctness
761	GetNodeValue	Calculate the result of multiplication operation

22.47.3 TFPMultiplyOperation.check

Synopsis: Validate the multiplication operation for correctness

Declaration: `procedure check;  Override`

Visibility: public

Description: The `check` method validates that both operands are numeric types suitable for multiplication, ensuring type compatibility.

22.47.4 TFPMultiplyOperation.AsString

Synopsis: Get string representation of multiplication operation

Declaration: `function AsString : AnsiString;  Override`

Visibility: public

Description: The `AsString` method returns a string representation showing both operands connected by the "*" symbol for debugging purposes.

22.47.5 TFPMultiplyOperation.GetNodeValue

Synopsis: Calculate the result of multiplication operation

Declaration: `procedure GetNodeValue(var Result: TFPExpressionResult);  Override`

Visibility: public

Description: The `GetNodeValue` method performs the actual multiplication by evaluating both operands and computing their product with appropriate type handling.

22.48 TFPNegateOperation

22.48.1 Description

The `TFPNegateOperation` class implements numeric negation operations that change the sign of numeric values. It extends TFPUnaryOperator (767) to provide negation-specific functionality for integers, floats, and currency values.

The negation operation converts positive values to negative and negative values to positive, essentially multiplying the operand by -1.

The operation requires that its operand be a numeric type (integer, float, or currency) and produces a result of the same or compatible numeric type.

This class is used internally by the TFPExpressionParser (728) class to construct an AST (Abstract Syntax Tree). There is normally no need for you to use or instantiate this class directly.

See also: TFPExprNode (750)

22.48.2 Method overview

Page	Method	Description
762	AsString	Get string representation of negation operation
761	Check	Validate the negation operation for correctness
762	GetNodeValue	Calculate the negated value
762	NodeType	Get the result type for negation operations

22.48.3 TFPNegateOperation.Check

Synopsis: Validate the negation operation for correctness

Declaration: `procedure Check;  Override`

Visibility: public

Description: The `Check` method validates that the operand of the negation operation is a numeric type suitable for negation. It ensures type compatibility and prevents attempts to negate non-numeric values.

The validation process checks that the operand is of a numeric type (integer, float, or currency) that supports negation operations.

22.48.4 TFPNegateOperation.NodeType

Synopsis: Get the result type for negation operations

Declaration: `function NodeType : TResultType;  Override`

Visibility: public

Description: The `NodeType` method returns the result type for negation operations, which typically matches the type of the operand. The result preserves the numeric type of the input (integer, float, or currency) to maintain type consistency.

22.48.5 TFPNegateOperation.GetNodeValue

Synopsis: Calculate the negated value

Declaration: `procedure GetNodeValue(var Result: TFPExpressionResult);  Override`

Visibility: public

Description: The `GetNodeValue` method performs the numeric negation by evaluating the operand and changing its sign. It multiplies the operand's value by -1, effectively converting positive values to negative and negative values to positive.

22.48.6 TFPNegateOperation.AsString

Synopsis: Get string representation of negation operation

Declaration: `function AsString : AnsiString;  Override`

Visibility: public

Description: The `AsString` method returns a string representation of the negation operation, typically showing the operand prefixed with a minus sign "-" for debugging and display purposes.

The string format helps in understanding the structure of arithmetic expressions and provides useful information for debugging expression parsing and evaluation.

22.49 TFPNotNode

22.49.1 Description

The `TFPNotNode` class implements the logical NOT operation that inverts a boolean value. It extends TFPUnaryOperator (767) to provide NOT-specific functionality that converts True to False and False to True.

The NOT operation requires that its operand evaluates to a boolean value. It performs the logical inversion of the boolean result, providing the fundamental logical negation operation for boolean expressions.

This class is used internally by the TFPExpressionParser (728) class to construct an AST (Abstract Syntax Tree). There is normally no need for you to use or instantiate this class directly.

See also: TFPExprNode (750)

22.49.2 Method overview

Page	Method	Description
763	AsString	Get string representation of NOT operation
763	Check	Validate the NOT operation for correctness
763	GetNodeValue	Calculate the result of NOT operation
763	NodeType	Get the result type for NOT operations

22.49.3 TFPNotNode.Check

Synopsis: Validate the NOT operation for correctness

Declaration: `procedure Check;  Override`

Visibility: public

Description: The `Check` method validates that the operand of the NOT operation is a boolean expression. It ensures type compatibility and prevents type mismatches that would result in runtime errors during evaluation.

The validation process checks that the operand produces a boolean result, raising appropriate errors if non-boolean operands are found.

22.49.4 TFPNotNode.NodeType

Synopsis: Get the result type for NOT operations

Declaration: `function NodeType : TResultType;  Override`

Visibility: public

Description: The NodeType (763) method returns rtBoolean (691) to indicate that NOT operations always produce boolean results. This type information is used by the expression parser for type checking and result handling.

22.49.5 TFPNotNode.GetNodeValue

Synopsis: Calculate the result of NOT operation

Declaration: `procedure GetNodeValue(var Result: TFPExpressionResult);  Override`

Visibility: public

Description: The `GetNodeValue` method performs the logical NOT operation by evaluating the operand and inverting its boolean result. It returns True if the operand is False, and False if the operand is True.

22.49.6 TFPNotNode.AsString

Synopsis: Get string representation of NOT operation

Declaration: `function AsString : AnsiString;  Override`

Visibility: public

Description: The `AsString` method returns a string representation of the NOT operation, typically showing the operand prefixed with the "not" keyword for debugging and display purposes.

22.50 TFPOrderingOperation

22.50.1 Description

The `TFPOrderingOperation` class is the base class for comparison operations that determine the ordering relationship between two values. It extends TFPBooleanResultOperation (721) to provide the foundation for less-than, greater-than, and related comparison operations.

The class handles the common aspects of ordering comparisons such as type checking and ensuring that the operands can be meaningfully compared. It provides the framework for operations that determine relative order rather than equality.

Derived classes implement specific ordering operations like <, >, <=, and >= while leveraging the common ordering infrastructure for validation and type management.

This class is used internally by the TFPExpressionParser (728) class to construct an AST (Abstract Syntax Tree). There is normally no need for you to use or instantiate this class directly.

See also: TFPExprNode (750)

22.50.2 Method overview

Page	Method	Description
764	Check	Validate the ordering operation for correctness

22.50.3 TFPOrderingOperation.Check

Synopsis: Validate the ordering operation for correctness

Declaration: `procedure Check;  Override`

Visibility: public

Description: The `Check` method validates that both operands of the ordering operation are compatible for comparison and can be meaningfully ordered. It ensures that the operands are of types that support ordering relationships.

The validation process checks that both operands are of comparable types (such as numeric types, strings, or dates) and that the comparison operation can be safely performed. It raises an exception if incompatible types are found.

The `Check` method extends the base validation from TFPBooleanResultOperation (721) with ordering-specific type checking requirements.

22.51 TFPPowerOperation

22.51.1 Description

The `TFPPowerOperation` class implements the power (exponentiation) operation for raising one numeric value to the power of another. It extends the mathematical operation framework to provide exponentiation functionality with proper type handling.

The power operation supports integer, float, and currency operands, automatically promoting the result type to ensure sufficient precision. The operation follows standard mathematical rules for exponentiation including handling of negative bases and fractional exponents.

This class is used internally by the TFPExpressionParser (728) class to construct an AST (Abstract Syntax Tree). There is normally no need for you to use or instantiate this class directly.

See also: TFPExprNode (750)

22.51.2 Method overview

Page	Method	Description
765	AsString	Get string representation of power operation
765	Check	Validate the power operation for correctness
766	GetNodeValue	Calculate the result of power operation
765	NodeType	Get the result type for power operations

22.51.3 TFPPowerOperation.Check

Synopsis: Validate the power operation for correctness

Declaration: `procedure Check;  Override`

Visibility: public

Description: The `Check` method validates that the power operation can be performed safely on the provided base and exponent operands. It checks for mathematically invalid operations such as negative bases with fractional exponents.

The validation also ensures that both operands are of compatible numeric types and that the operation will not result in overflow or underflow conditions that could produce invalid results.

22.51.4 TFPPowerOperation.AsString

Synopsis: Get string representation of power operation

Declaration: `function AsString : AnsiString;  Override`

Visibility: public

Description: The AsString (765) method returns a human-readable string representation of the power operation, typically in the format 'base êxponent' or 'base ** exponent', showing both operands and the power operator.

22.51.5 TFPPowerOperation.NodeType

Synopsis: Get the result type for power operations

Declaration: `function NodeType : TResultType;  Override`

Visibility: public

Description: The `NodeType` method determines the data type that will be produced by the power operation. Power operations typically promote the result to a floating-point type to handle fractional results and maintain mathematical precision.

The result type is usually float regardless of the input operand types, since exponentiation often produces non-integer results even when both operands are integers.

22.51.6 TFPPowerOperation.GetNodeValue

Synopsis: Calculate the result of power operation

Declaration: `procedure GetNodeValue(var Result: TFPExpressionResult);  Override`

Visibility: public

Description: The GetNodeValue (766) method evaluates the power operation by raising the left operand to the power of the right operand and returns the result. It handles both integer and floating-point base values with numeric exponents.

22.52 TFPSubtractOperation

22.52.1 Description

The `TFPSubtractOperation` class implements the subtraction operation between two numeric operands. It extends the binary operation framework to provide subtraction-specific functionality for integers, floats, and currency values.

The operation handles type conversion automatically when operands have different numeric types, promoting the result to the more precise type as needed. The result type follows standard numeric promotion rules.

The subtraction operation computes the difference between the left operand (minuend) and the right operand (subtrahend), returning left - right.

This class is used internally by the TFPExpressionParser (728) class to construct an AST (Abstract Syntax Tree). There is normally no need for you to use or instantiate this class directly.

See also: TFPExprNode (750)

22.52.2 Method overview

Page	Method	Description
767	AsString	Get string representation of subtraction operation
766	Check	Validate the subtraction operation for correctness
766	GetNodeValue	Calculate the result of subtraction operation

22.52.3 TFPSubtractOperation.Check

Synopsis: Validate the subtraction operation for correctness

Declaration: `procedure Check;  Override`

Visibility: public

Description: The Check (766) method validates that both operands are numeric types suitable for subtraction. It ensures type compatibility and prevents attempts to subtract incompatible types.

22.52.4 TFPSubtractOperation.GetNodeValue

Synopsis: Calculate the result of subtraction operation

Declaration: `procedure GetNodeValue(var Result: TFPExpressionResult);  Override`

Visibility: public

Description: The GetNodeValue (766) method performs the actual subtraction by evaluating both operands and computing their difference. It handles type conversion and returns the result in the appropriate numeric format.

22.52.5 TFPSubtractOperation.AsString

Synopsis: Get string representation of subtraction operation

Declaration: `function AsString : AnsiString;  Override`

Visibility: public

Description: The `AsString` method returns a string representation of the subtraction operation, showing both operands connected by the "-" symbol for debugging and display purposes.

22.53 TFPUnaryOperator

22.53.1 Description

`TFPUnaryOperator` is the base class for all unary operations in the expression parser. Unary operations take a single operand and perform some operation to produce a result. It extends TFPExprNode (750) to provide the foundation for all single-operand operations.

The class manages the operand node and provides common functionality such as aggregate handling, validation, and resource management that is shared by all unary operations.

Derived classes implement specific operations such as logical NOT, numeric negation, and type conversions. Each operation defines its own evaluation logic while leveraging the common unary operation infrastructure.

This class is used internally by the TFPExpressionParser (728) class to construct an AST (Abstract Syntax Tree). There is normally no need for you to use or instantiate this class directly.

See also: TFPExprNode (750)

22.53.2 Method overview

Page	Method	Description
769	Check	Validate the unary operator for correctness
767	Create	Create unary operator with operand
768	Destroy	Destroy unary operator and free resources
769	HasAggregate	Check if unary operator contains aggregates
768	InitAggregate	Initialize aggregate processing for unary operator
768	UpdateAggregate	Update aggregate calculations for unary operator

22.53.3 Property overview

Page	Properties	Access	Description
769	Operand	r	The operand expression of the unary operator

22.53.4 TFPUnaryOperator.Create

Synopsis: Create unary operator with operand

Declaration: `constructor Create(AOperand: TFPExprNode)`

Visibility: public

Description: The TFPUnaryOperator (767) constructor creates a new unary operation with the specified operand. It takes ownership of the operand node and manages its lifecycle as part of the unary operation.

The constructor properly initializes the unary operation structure and establishes the relationship between the operation and its operand, ensuring that the operand is properly managed during the operation's lifetime.

22.53.5 TFPUnaryOperator.Destroy

Synopsis: Destroy unary operator and free resources

Declaration: `destructor Destroy;  Override`

Visibility: public

Description: This destructor properly cleans up the unary operation by freeing the operand node and any other resources used by the operation.

The destructor ensures that all memory allocated for the operand is properly released, preventing memory leaks in complex expression trees.

22.53.6 TFPUnaryOperator.InitAggregate

Synopsis: Initialize aggregate processing for unary operator

Declaration: `procedure InitAggregate;  Override`

Visibility: public

Description: The `InitAggregate` method initializes aggregate processing for the operand of the unary operation. It delegates the aggregate initialization to the operand to ensure that any aggregate expressions within the operand are properly initialized.

The `InitAggregate` method is called as part of the aggregate processing cycle when the expression contains aggregate functions that need special handling.

22.53.7 TFPUnaryOperator.UpdateAggregate

Synopsis: Update aggregate calculations for unary operator

Declaration: `procedure UpdateAggregate;  Override`

Visibility: public

Description: The `UpdateAggregate` method updates aggregate calculations for the operand of the unary operation. It delegates the aggregate updates to the operand to ensure that any aggregate expressions are properly maintained during the aggregation cycle.

The method is called repeatedly during aggregate processing to update the state of aggregate functions contained within the operand.

22.53.8 TFPUnaryOperator.HasAggregate

Synopsis: Check if unary operator contains aggregates

Declaration: `function HasAggregate : Boolean;  Override`

Visibility: public

Description: The `HasAggregate` method checks whether the operand contains aggregate expressions. It returns True if the operand has aggregate functionality, which indicates that special aggregate processing is required for this operation.

The method is used by the expression parser to identify operations that need aggregate handling and to ensure proper processing order for aggregate expressions.

22.53.9 TFPUnaryOperator.Check

Synopsis: Validate the unary operator for correctness

Declaration: `procedure Check;  Override`

Visibility: public

Description: The `Check` method validates the unary operation by checking the operand for correctness and ensuring that it is compatible with the specific operation. It calls the `Check` method on the operand to perform recursive validation.

22.53.10 TFPUnaryOperator.Operand

Synopsis: The operand expression of the unary operator

Declaration: `Property Operand : TFPExprNode`

Visibility: public

Access: Read

Description: The Operand (769) property provides access to the operand of the unary operation. The operand is the single expression node that the unary operation acts upon to produce its result.

The property returns the TFPExprNode (750) that represents the operand, allowing access to its value and type information for evaluation purposes.

22.54 TFPUnequalOperation

22.54.1 Description

The `TFPUnequalOperation` class implements the inequality comparison operation (<>) between two operands of compatible types. It extends TFPBooleanResultOperation (721) to provide inequality-specific comparison logic.

The operation compares two values and returns True if they are not equal, False if they are equal. It handles different data types appropriately, using numeric comparison for numbers, string comparison for strings, and boolean comparison for boolean values.

Type conversion is performed when necessary to compare values of different but compatible types, ensuring that the inequality test works across different numeric types.

See also: TFPExprNode (750)

22.54.2 Method overview

Page	Method	Description
770	AsString	Get string representation of inequality operation

22.54.3 TFPUnequalOperation.AsString

Synopsis: Get string representation of inequality operation

Declaration: `function AsString : AnsiString;  Override`

Visibility: public

Description: The `AsString` method returns a string representation of the inequality operation, showing both operands connected by the "<>" for debugging and display purposes.

The string format helps in understanding the structure of complex comparison expressions and provides useful information for debugging expression parsing and evaluation.

22.55 TIfOperation

22.55.1 Description

The `TIfOperation` class implements conditional if-then-else operations in expressions, allowing for branching logic based on boolean conditions. It extends the expression framework to provide conditional evaluation similar to the ternary operator in many programming languages.

The operation evaluates a boolean condition and returns one of two expressions based on the result: the "then" expression if the condition is True, or the "else" expression if the condition is False. This enables complex conditional logic within expressions.

The class manages three operands: the condition (which must evaluate to boolean), the "then" expression (evaluated when condition is True), and the "else" expression (evaluated when condition is False). Only one of the result expressions is evaluated, providing efficient conditional processing.

This class is used internally by the TFPExpressionParser (728) class to construct an AST (Abstract Syntax Tree). There is normally no need for you to use or instantiate this class directly.

See also: TFPExprNode (750)

22.55.2 Method overview

Page	Method	Description
772	AsString	Get string representation of conditional operation
771	Check	Validate the conditional operation for correctness
772	Create	Create conditional operation with condition and branches
772	destroy	Destroy conditional operation and free resources
771	HasAggregate	Check if conditional operation contains aggregates
771	InitAggregate	Initialize aggregate processing for if operation
772	NodeType	Get the result type for conditional operation
771	UpdateAggregate	Update aggregate calculations for if operation

22.55.3 Property overview

Page	Properties	Access	Description
773	Condition	r	The condition expression of the if operation

22.55.4 TIfOperation.Check

Synopsis: Validate the conditional operation for correctness

Declaration: `procedure Check;  Override`

Visibility: public

Description: The `Check` method validates the conditional operation by checking that the condition expression evaluates to a boolean type and that the then-branch and else-branch expressions are present and valid.

The validation ensures that the condition is a boolean expression and that both branch expressions are compatible for the intended result type. It also validates all three operand expressions recursively.

22.55.5 TIfOperation.InitAggregate

Synopsis: Initialize aggregate processing for if operation

Declaration: `procedure InitAggregate;  Override`

Visibility: public

Description: The `InitAggregate` method initializes aggregate processing for the condition, then-branch, and else-branch expressions of the conditional operation. It delegates the aggregate initialization to all three operands.

The InitAggregate (771) method is called as part of the aggregate processing cycle when any of the conditional branches contain aggregate functions that need special handling.

22.55.6 TIfOperation.UpdateAggregate

Synopsis: Update aggregate calculations for if operation

Declaration: `procedure UpdateAggregate;  Override`

Visibility: public

Description: The `UpdateAggregate` method updates aggregate calculations for the condition, then-branch, and else-branch expressions of the conditional operation. It delegates the aggregate updates to all three operands.

The method is called repeatedly during aggregate processing to update the state of any aggregate functions contained within the conditional expressions.

22.55.7 TIfOperation.HasAggregate

Synopsis: Check if conditional operation contains aggregates

Declaration: `function HasAggregate : Boolean;  Override`

Visibility: public

Description: The `HasAggregate` method checks whether any of the three operands (condition, then-branch, or else-branch) contain aggregate expressions. I t returns True if any operand has aggregate functionality.

The method is used by the expression parser to identify conditional operations that need aggregate handling and to ensure proper processing order for expressions with aggregate functions.

22.55.8 TIfOperation.NodeType

Synopsis: Get the result type for conditional operation

Declaration: `function NodeType : TResultType;  Override`

Visibility: public

Description: The `NodeType` method determines and returns the result type for the conditional operation based on the types of the then-branch and else-branch expressions. The result type is typically the common type that both branches can be converted to.

The method analyzes both branch expressions to determine a compatible result type that can accommodate values from either branch, ensuring type safety in conditional expressions.

22.55.9 TIfOperation.Create

Synopsis: Create conditional operation with condition and branches

Declaration:
```
constructor Create(ACondition: TFPExprNode; ALeft: TFPExprNode;
                   ARight: TFPExprNode)
```

Visibility: public

Description: The `Create` constructor creates a new conditional operation with the specified condition expression, then-branch expression, and else-branch expression. It takes ownership of all three operand nodes.

The constructor properly initializes the conditional operation structure and establishes the relationship between the operation and its three operands (condition, then-branch, and else-branch), ensuring proper resource management.

22.55.10 TIfOperation.destroy

Synopsis: Destroy conditional operation and free resources

Declaration: `destructor destroy;  Override`

Visibility: public

Description: `Destroy` properly cleans up the conditional operation by freeing the condition expression, then-branch expression, and else-branch expression, along with any other resources used by the operation.

22.55.11 TIfOperation.AsString

Synopsis: Get string representation of conditional operation

Declaration: `function AsString : AnsiString;  Override`

Visibility: public

Description: The `AsString` method returns a string representation of the conditional operation, typically showing the condition, then-branch, and else-branch expressions in a readable format such as "if condition then expr1 else expr2" for debugging and display purposes.

22.55.12 TIfOperation.Condition

Synopsis: The condition expression of the if operation

Declaration: `Property Condition : TFPExprNode`

Visibility: public

Access: Read

Description: The `Condition` property provides access to the boolean condition expression that determines which branch of the conditional operation will be evaluated. The condition must evaluate to a boolean result.

The condition is evaluated first, and based on its boolean result, either the then-branch or else-branch expression is evaluated to provide the final result of the conditional operation.

22.56 TIntConvertNode

22.56.1 Description

The `TIntConvertNode` class handles conversion operations that produce integer results from various source data types. It extends TFPConvertNode (726) to provide integer-specific conversion functionality.

The class is used when expressions require values to be converted to integer representation, such as when performing integer arithmetic operations or when explicit integer casting is needed.

The conversion handles different source types appropriately, applying truncation, rounding, or other integer conversion rules based on the source data type.

This class is used internally by the TFPExpressionParser (728) class to construct an AST (Abstract Syntax Tree). There is normally no need for you to use or instantiate this class directly.

See also: TFPExprNode (750)

22.56.2 Method overview

Page	Method	Description
773	Check	Validate the integer conversion for correctness

22.56.3 TIntConvertNode.Check

Synopsis: Validate the integer conversion for correctness

Declaration: `procedure Check;  Override`

Visibility: public

Description: The `Check` method validates that the conversion operation can be safely performed and that the source value can be properly converted to an integer. It checks for potential overflow conditions and invalid source data types.

The validation helps prevent runtime errors during expression evaluation by ensuring that all type conversions are valid before the actual conversion is attempted.

22.57 TIntToCurrencyNode

22.57.1 Description

This conversion node class handles the automatic conversion of integer values to currency values in expressions. It extends the base conversion functionality to provide integer-to-currency specific conversion logic.

The class is used internally by the expression parser when type coercion is needed to make integer values compatible with currency operations. The conversion treats the integer as a currency amount with appropriate decimal precision.

This conversion is useful in financial calculations where integer values need to be combined with currency values while maintaining proper decimal precision and formatting.

See also: TFPExprNode (750)

22.57.2 Method overview

Page	Method	Description
774	GetNodeValue	Convert integer value to currency
774	NodeType	Get the result type for integer to currency conversion

22.57.3 TIntToCurrencyNode.NodeType

Synopsis: Get the result type for integer to currency conversion

Declaration: `function NodeType : TResultType;  Override`

Visibility: public

Description: The NodeType (774) method returns rtCurrency (691) to indicate that this conversion node produces currency results. It overrides the base class method to provide type-specific result type information.

The method enables the expression parser to properly identify currency operations and apply appropriate formatting, precision, and arithmetic rules for currency calculations.

This type information is crucial for maintaining proper currency handling throughout complex expressions involving mixed numeric types.

22.57.4 TIntToCurrencyNode.GetNodeValue

Synopsis: Convert integer value to currency

Declaration: `procedure GetNodeValue(var Result: TFPExpressionResult);  Override`

Visibility: public

Description: The `GetNodeValue` method performs the actual conversion of an integer value to a currency value. It retrieves the integer value from the source node and converts it to the equivalent currency representation.

The conversion maintains the numeric value while applying currency-specific precision and formatting rules. The integer is treated as a whole currency unit (e.g., dollars, euros) with appropriate decimal handling.

The result is returned in a TFPExpressionResult (695) structure with the result type set to rtCurrency and the converted value stored in the currency field.

22.58 TIntToDateTimeNode

22.58.1 Description

The `TIntToDateTimeNode` conversion node class handles the conversion of integer values to date/time values in expressions. It extends the base conversion functionality to provide integer-to-datetime specific conversion logic.

The class interprets integer values as datetime representations, typically using the standard Pascal datetime encoding where integers represent days since a base date. This allows integration of numeric date calculations with datetime operations.

The conversion enables mathematical operations on dates by treating them as numeric values while maintaining proper datetime semantics and formatting in the final results.

See also: TFPExprNode (750)

22.58.2 Method overview

Page	Method	Description
775	GetNodeValue	Internally used convert integer value to datetime
775	NodeType	Get the result type for integer to datetime conversion

22.58.3 TIntToDateTimeNode.NodeType

Synopsis: Get the result type for integer to datetime conversion

Declaration: `function NodeType : TResultType;  Override`

Visibility: public

Description: The `NodeType` method returns rtDateTime (691) to indicate that this conversion node produces datetime results. It provides type information for the expression parser to handle datetime operations correctly.

22.58.4 TIntToDateTimeNode.GetNodeValue

Synopsis: Internally used convert integer value to datetime

Declaration: `procedure GetNodeValue(var Result: TFPExpressionResult);  Override`

Visibility: public

Description: The `GetNodeValue` method converts an integer value to a datetime value using the standard Pascal datetime encoding. The integer is interpreted as the number of days since the base date (typically December 30, 1899).

See also: TFPExprNode (750)

22.59 TIntToFloatNode

22.59.1 Description

The `TIntToFloatNode` conversion node class handles the automatic conversion of integer values to floating-point values in expressions. It extends the base conversion functionality to provide integer-to-float specific conversion logic.

The class is used internally by the expression parser when type coercion is needed to make integer values compatible with operations that require floating-point operands. The conversion preserves the numeric value while changing the data type representation.

This class is used internally by the TFPExpressionParser (728) class to construct an AST (Abstract Syntax Tree). There is normally no need for you to use or instantiate this class directly.

See also: TFPExprNode (750)

22.59.2 Method overview

Page	Method	Description
776	GetNodeValue	Internally used convert integer value to float
776	NodeType	Get the result type for integer to float conversion

22.59.3 TIntToFloatNode.NodeType

Synopsis: Get the result type for integer to float conversion

Declaration: `function NodeType : TResultType;  Override`

Visibility: public

Description: The `NodeType` method returns rtFloat (691) to indicate that this conversion node produces floating-point results. It overrides the base class method to provide type-specific result type information.

The method is used by the expression parser to determine the result type of expressions that include this conversion node, enabling proper type checking and result handling.

22.59.4 TIntToFloatNode.GetNodeValue

Synopsis: Internally used convert integer value to float

Declaration: `procedure GetNodeValue(var Result: TFPExpressionResult);  Override`

Visibility: public

Description: The `GetNodeValue` method performs the actual conversion of an integer value to a floating-point value. It retrieves the integer value from the source node and converts it to the equivalent floating-point representation.

The conversion maintains the numeric value while changing the internal representation to TExprFloat (688) format. The method ensures that the conversion is performed safely and accurately.

The result is returned in a TFPExpressionResult (695) structure with the result type set to rtFloat and the converted value stored in the appropriate field.

See also: TFPExprNode (750)

22.60 TMathOperation

22.60.1 Description

The TMathOperation (776) class serves as a base class for mathematical operations that can be performed on numeric operands. It provides common functionality for arithmetic operations including type checking and result type determination.

Mathematical operations support various numeric types including integers, floats, and currency values, with automatic type promotion to ensure precision is maintained in calculations. The class ensures that operands are compatible for mathematical operations.

This class is used internally by the TFPExpressionParser (728) class to construct an AST (Abstract Syntax Tree). There is normally no need for you to use or instantiate this class directly.

See also: TFPExprNode (750)

22.60.2 Method overview

Page	Method	Description
777	Check	Validate the mathematical operation for correctness
777	NodeType	Get the result type for mathematical operations

22.60.3 TMathOperation.Check

Synopsis: Validate the mathematical operation for correctness

Declaration: `procedure Check;  Override`

Visibility: public

Description: The `Check` method validates that the mathematical operation can be performed safely on the provided operands. It checks that both operands are of compatible numeric types and that the operation is mathematically valid.

The validation includes type compatibility checking and ensures that operations like division by zero or other mathematically undefined operations are detected before evaluation attempts.

22.60.4 TMathOperation.NodeType

Synopsis: Get the result type for mathematical operations

Declaration: `function NodeType : TResultType;  Override`

Visibility: public

Description: The `NodeType` method determines the data type that will be produced by the mathematical operation. The result type is calculated based on the types of the operands and follows standard type promotion rules.

For example, operations between integers produce integer results, but operations involving floats promote the result to float type to maintain precision. Currency operations maintain currency type where appropriate.

Chapter 23

Reference for unit 'fphttp'

23.1 Used units

Table 23.1: Used units by unit 'fphttp'

Name	Page
Classes	??
HTTPDefs	1141
httproute	1225
System	??
sysutils	??

23.2 Overview

The `fphttp` unit provides the foundation for handling HTTP requests in web server applications. It implements content producers (787), web actions (783), and web modules (781) for processing HTTP requests and generating responses.

The idea behind the web module and action architecture is that they process an URL of the following kind:

```
{BaseURL}/module/action/*
```

Where the appropriate module is determined by the http router (778) and the correct action is selected by the module. The action's event handler will then be executed.

If your application requires a more flexible URL handling, you may want to register callbacks directly with the HTTP router. If you are looking to generate HTML, then the fpweb (778) unit contains web modules that descend from TCustomHTTPModule (781) but are geared towards generating HTML.

23.3 Constants, types and variables

23.3.1 Resource strings

```
SErrInvActNoDefaultAction =
  'Invalid action name and no default action'
```

SErrInvActNoDefaultAction is the error message displayed when an invalid action is requested and no default action is available.

```
SErrNoDefaultAction = 'No action name and no default action'
```

SErrNoDefaultAction is the error message displayed when no default action is available.

```
SErrNoSessionFactoryClass =
  'No session manager class available. Include iniwebsession unit and recompile.'
```

SErrNoSessionFactoryClass is the error message displayed when no session factory class is registered.

```
SErrNoSessionOutsideRequest =
  'Default session not available outside handlerequest'
```

SErrNoSessionOutsideRequest is the error message displayed when attempting to access session data outside of a request context.

```
SErrNoSuchAction = 'No action found for action: "%s"'
```

SErrNoSuchAction is the error message displayed when a requested action cannot be found.

```
SErrNosuchModule = 'No such module registered: "%s"'
```

SErrNosuchModule is the error message displayed when a requested module cannot be found.

```
SErrRequestNotHandled = 'Web request was not handled by actions.'
```

SErrRequestNotHandled is the error message displayed when a request cannot be handled by any action.

```
SErrUnknownAction = 'Unknown action: "%s"'
```

SErrUnknownAction is the error message displayed for unrecognized or invalid actions.

23.3.2 Types

```
TCustomHTTPModuleClass = Class of TCustomHTTPModule
```

TCustomHTTPModuleClass is a class reference type for HTTP module classes.

```
TGetActionEvent = procedure(Sender: TObject; ARequest: TRequest;
  var ActionName: string) of object
```

TGetActionEvent is triggered to determine which action should handle a specific request.

```
TInitModuleEvent = procedure(Sender: TObject; ARequest: TRequest)
  of object
```

TInitModuleEvent is triggered when an HTTP module is initialized for request processing.

```
TOnModuleRequest = procedure(Sender: TModuleItem; ARequest: TRequest
  ;
                             AResponse: TResponse) of object
```

`TOnModuleRequest` is an event type for handling requests at the module level.

```
TSessionFactoryClass = Class of TSessionFactory
```

`TSessionFactoryClass` is a class reference type for session factory classes.

```
TSessionHTTPModuleClass = Class of TSessionHTTPModule
```

`TSessionHTTPModuleClass` is a class reference type for session-enabled HTTP module classes.

```
TWebActionEvent = procedure(Sender: TObject; ARequest: TRequest;
  AResponse: TResponse; var Handled: Boolean
  )
                             of object
```

`TWebActionEvent` is an event type for handling web action processing before or after request handling.

```
TWebModuleKind = (wkPooled,wkOneShot)
```

Table 23.2: Enumeration values for type TWebModuleKind

Value	Explanation
wkOneShot	New module instance created for each request
wkPooled	Module instances are pooled and reused

`TWebModuleKind` defines how web modules are created and managed during request handling.

23.3.3 Variables

```
ModuleFactory : TModuleFactory
```

`ModuleFactory` is the global instance of the module factory for managing HTTP modules.

```
SessionFactoryClass : TSessionFactoryClass = Nil
```

`SessionFactoryClass` holds the class reference used to create session factory instances.

23.4 Procedures and functions

23.4.1 RegisterHTTPModule

Synopsis: Registers an HTTP module with the factory

Declaration:
```
procedure RegisterHTTPModule(ModuleClass: TCustomHTTPModuleClass;
                             SkipStreaming: Boolean)
procedure RegisterHTTPModule(const ModuleName: string;
                             ModuleClass: TCustomHTTPModuleClass;
                             SkipStreaming: Boolean)
```

Visibility: default

Description: `RegisterHTTPModule` is a convenience function to register an HTTP module class with the global module factory.

23.4.2 SessionFactory

Synopsis: Returns the global session factory instance

Declaration:
```
function SessionFactory : TSessionFactory
```

Visibility: default

Description: `SessionFactory` returns the global session factory instance used for session management.

23.5 EFPHTTPError

23.5.1 Description

`EFPHTTPError` is raised when errors occur during HTTP request processing.

23.6 EWebSessionError

23.6.1 Description

`EWebSessionError` is raised when errors occur during session management operations.

23.7 TCustomHTTPModule

23.7.1 Description

`TCustomHTTPModule` is the base class for HTTP modules that handle web requests. It provides the foundation for web applications with action-based request routing.

23.7.2 Method overview

Page	Method	Description
782	CreateNew	Creates a new HTTP module instance
782	DoAfterInitModule	Executes after module initialization
782	HandleRequest	Handles an HTTP request
782	RegisterModule	Registers the module with the factory

23.7.3 Property overview

Page	Properties	Access	Description
783	AfterInitModule	rw	Event after module initialization
783	BaseURL	rw	Base URL for the module
783	Kind	rw	Module creation mode

23.7.4 TCustomHTTPModule.CreateNew

Synopsis: Creates a new HTTP module instance

Declaration:
```
constructor CreateNew(aOwner: TComponent; CreateMode: Integer)
                          ; Override; Overload
```

Visibility: public

Description: `CreateNew` creates a new HTTP module instance with the specified owner and creation mode.

23.7.5 TCustomHTTPModule.RegisterModule

Synopsis: Registers the module with the factory

Declaration:
```
class procedure RegisterModule(const AModuleName: string); Overload
class procedure RegisterModule(const AModuleName: string;
                               ASkipStreaming: Boolean); Overload
```

Visibility: public

Description: `RegisterModule` registers this HTTP module class with the module factory using the specified name.

23.7.6 TCustomHTTPModule.HandleRequest

Synopsis: Handles an HTTP request

Declaration:
```
procedure HandleRequest(ARequest: TRequest; AResponse: TResponse)
                        ; Virtual; Abstract
```

Visibility: public

Description: `HandleRequest` processes an HTTP request and generates the appropriate response.

23.7.7 TCustomHTTPModule.DoAfterInitModule

Synopsis: Executes after module initialization

Declaration:
```
procedure DoAfterInitModule(ARequest: TRequest); Virtual
```

Visibility: public

Description: `DoAfterInitModule` is called after the module has been initialized for the current request.

23.7.8 TCustomHTTPModule.Kind

Synopsis: Module creation mode

Declaration: `Property Kind : TWebModuleKind`

Visibility: public

Access: Read,Write

Description: `Kind` determines how module instances are created and managed.

23.7.9 TCustomHTTPModule.BaseURL

Synopsis: Base URL for the module

Declaration: `Property BaseURL : string`

Visibility: public

Access: Read,Write

Description: `BaseURL` specifies the base URL path that this module handles.

23.7.10 TCustomHTTPModule.AfterInitModule

Synopsis: Event after module initialization

Declaration: `Property AfterInitModule : TInitModuleEvent`

Visibility: public

Access: Read,Write

Description: `AfterInitModule` is triggered after the module is initialized for request processing.

23.8 TCustomWebAction

23.8.1 Description

`TCustomWebAction` represents an individual web action that can handle specific HTTP requests. It associates a content producer or a simple event with request handling events. Web actions are selected by the TCustomHTTPModule (781) module by checking the second path element in the HTTP URL:

```
{BaseURL}/module/action
```

23.8.2 Method overview

Page	Method	Description
784	GetNamePath	Returns the action name path

23.8.3 Property overview

Page	Properties	Access	Description
785	AfterResponse	rw	Event triggered after response generation
785	BeforeRequest	rw	Event triggered before request processing
784	ContentProducer	rw	Content producer for this action
784	Default	rw	Indicates if this is the default action
784	Name	rw	Name of the web action

23.8.4 TCustomWebAction.GetNamePath

Synopsis: Returns the action name path

Declaration: `function GetNamePath : string;  Override`

Visibility: public

Description: `GetNamePath` returns the full path name of the action for identification purposes.

23.8.5 TCustomWebAction.Name

Synopsis: Name of the web action

Declaration: `Property Name : string`

Visibility: published

Access: Read,Write

Description: `Name` specifies the unique name used to identify this web action.

23.8.6 TCustomWebAction.ContentProducer

Synopsis: Content producer for this action

Declaration: `Property ContentProducer : THTTPContentProducer`

Visibility: published

Access: Read,Write

Description: `ContentProducer` specifies the content producer responsible for generating the response content.

23.8.7 TCustomWebAction.Default

Synopsis: Indicates if this is the default action

Declaration: `Property Default : Boolean`

Visibility: published

Access: Read,Write

Description: `Default` specifies whether this action should be used as the default when no specific action is requested.

23.8.8 TCustomWebAction.BeforeRequest

Synopsis: Event triggered before request processing

Declaration: `Property BeforeRequest : TRequestEvent`

Visibility: published

Access: Read,Write

Description: `BeforeRequest` is triggered before the request is processed by the action.

23.8.9 TCustomWebAction.AfterResponse

Synopsis: Event triggered after response generation

Declaration: `Property AfterResponse : TResponseEvent`

Visibility: published

Access: Read,Write

Description: `AfterResponse` is triggered after the response has been generated by the action.

23.9 TCustomWebActions

23.9.1 Description

`TCustomWebActions` manages a collection of web actions that can handle HTTP requests.

23.9.2 Method overview

Page	Method	Description
786	ActionByName	Finds action by name
786	Add	Adds a new web action
786	Assign	Copies actions from another collection
785	Create	Creates a new web actions collection
786	FindAction	Searches for action by name
786	IndexOfAction	Returns index of named action

23.9.3 Property overview

Page	Properties	Access	Description
787	Actions	rw	Array access to web actions
787	DefActionWhenUnknown	rw	Use default action for unknown requests
786	OnGetAction	rw	Event for action name determination

23.9.4 TCustomWebActions.Create

Synopsis: Creates a new web actions collection

Declaration: `constructor Create(AItemClass: TCollectionItemClass)`

Visibility: public

Description: `Create` initializes a new web actions collection with the specified action item class.

23.9.5 TCustomWebActions.Assign

Synopsis: Copies actions from another collection

Declaration: `procedure Assign(Source: TPersistent);  Override`

Visibility: public

Description: `Assign` copies all actions from the source collection to this collection.

23.9.6 TCustomWebActions.Add

Synopsis: Adds a new web action

Declaration: `function Add : TCustomWebAction`

Visibility: public

Description: `Add` creates and adds a new web action to the collection.

23.9.7 TCustomWebActions.ActionByName

Synopsis: Finds action by name

Declaration: `function ActionByName(const AName: string) : TCustomWebAction`

Visibility: public

Description: `ActionByName` returns the web action with the specified name, or raises an exception if not found.

23.9.8 TCustomWebActions.FindAction

Synopsis: Searches for action by name

Declaration: `function FindAction(const AName: string) : TCustomWebAction`

Visibility: public

Description: `FindAction` searches for a web action with the specified name and returns nil if not found.

23.9.9 TCustomWebActions.IndexOfAction

Synopsis: Returns index of named action

Declaration: `function IndexOfAction(const AName: string) : Integer`

Visibility: public

Description: `IndexOfAction` returns the index of the web action with the specified name, or -1 if not found.

23.9.10 TCustomWebActions.OnGetAction

Synopsis: Event for action name determination

Declaration: `Property OnGetAction : TGetActionEvent`

Visibility: public

Access: Read,Write

Description: `OnGetAction` is triggered to determine which action should handle a request.

23.9.11 TCustomWebActions.Actions

Synopsis: Array access to web actions

Declaration: `Property Actions[Index: Integer]: TCustomWebAction; default`

Visibility: public

Access: Read,Write

Description: `Actions` provides indexed access to individual web actions in the collection.

23.9.12 TCustomWebActions.DefActionWhenUnknown

Synopsis: Use default action for unknown requests

Declaration: `Property DefActionWhenUnknown : Boolean`

Visibility: public

Access: Read,Write

Description: `DefActionWhenUnknown` determines whether the default action should handle requests for unknown actions.

23.10 THTTPContentProducer

23.10.1 Description

`THTTPContentProducer` is the base class for generating HTTP content. It provides methods for handling requests and producing content for HTTP responses.

23.10.2 Method overview

Page	Method	Description
788	ContentToStream	Writes content to a stream
788	GetContent	Retrieves content for the HTTP response
787	HandleRequest	Handles an HTTP request
788	HaveContent	Checks if content is available

23.10.3 Property overview

Page	Properties	Access	Description
788	Request	r	Current HTTP request
788	Response	r	Current HTTP response

23.10.4 THTTPContentProducer.HandleRequest

Synopsis: Handles an HTTP request

Declaration:
```
procedure HandleRequest(ARequest: TRequest; AResponse: TResponse;
                        var Handled: Boolean)
```

Visibility: public

Description: `HandleRequest` processes an HTTP request and sets the handled flag if the request was processed.

23.10.5 THTTPContentProducer.GetContent

Synopsis: Retrieves content for the HTTP response

Declaration:
```
procedure GetContent(ARequest: TRequest; Content: TStream;
                     var Handled: Boolean)
```

Visibility: public

Description: `GetContent` retrieves the content to be sent as the HTTP response body.

23.10.6 THTTPContentProducer.HaveContent

Synopsis: Checks if content is available

Declaration:
```
function HaveContent : Boolean;  Virtual
```

Visibility: public

Description: `HaveContent` returns `True` if the content producer can generate content for the current request.

23.10.7 THTTPContentProducer.ContentToStream

Synopsis: Writes content to a stream

Declaration:
```
function ContentToStream(Stream: TStream) : Boolean;  Virtual
```

Visibility: public

Description: `ContentToStream` writes the generated content directly to a stream and returns the number of bytes written.

23.10.8 THTTPContentProducer.Request

Synopsis: Current HTTP request

Declaration:
```
Property Request : TRequest
```

Visibility: public

Access: Read

Description: `Request` provides access to the current HTTP request being processed.

23.10.9 THTTPContentProducer.Response

Synopsis: Current HTTP response

Declaration:
```
Property Response : TResponse
```

Visibility: public

Access: Read

Description: `Response` provides access to the current HTTP response being generated.

23.11 TModuleFactory

23.11.1 Description

`TModuleFactory` manages the registration and creation of HTTP modules.

23.11.2 Method overview

Page	Method	Description
789	FindModule	Finds a module by name
790	IndexOfModule	Returns index of named module
789	ModuleByName	Returns module by name
789	RegisterHTTPModule	Registers an HTTP module

23.11.3 Property overview

Page	Properties	Access	Description
790	Modules	rw	Array access to registered modules
790	OnModuleRequest	rw	Event for module request handling

23.11.4 TModuleFactory.RegisterHTTPModule

Synopsis: Registers an HTTP module

Declaration:
```
procedure RegisterHTTPModule(const ModuleName: string;
                             ModuleClass: TCustomHTTPModuleClass;
                             SkipStreaming: Boolean);  Virtual
procedure RegisterHTTPModule(ModuleClass: TCustomHTTPModuleClass;
                             SkipStreaming: Boolean)
```

Visibility: public

Description: `RegisterHTTPModule` registers an HTTP module class with the factory.

23.11.5 TModuleFactory.FindModule

Synopsis: Finds a module by name

Declaration:
```
function FindModule(const AModuleName: string) : TModuleItem
```

Visibility: public

Description: `FindModule` searches for a registered module with the specified name.

23.11.6 TModuleFactory.ModuleByName

Synopsis: Returns module by name

Declaration:
```
function ModuleByName(const AModuleName: string) : TModuleItem
```

Visibility: public

Description: `ModuleByName` returns the module item with the specified name, or raises an exception if not found.

23.11.7 TModuleFactory.IndexOfModule

Synopsis: Returns index of named module

Declaration: `function IndexOfModule(const AModuleName: string) : Integer`

Visibility: public

Description: `IndexOfModule` returns the index of the module with the specified name, or -1 if not found.

23.11.8 TModuleFactory.Modules

Synopsis: Array access to registered modules

Declaration: `Property Modules[Index: Integer]: TModuleItem; default`

Visibility: public

Access: Read,Write

Description: `Modules` provides indexed access to registered module items.

23.11.9 TModuleFactory.OnModuleRequest

Synopsis: Event for module request handling

Declaration: `Property OnModuleRequest : TOnModuleRequest`

Visibility: public

Access: Read,Write

Description: `OnModuleRequest` is triggered when a request is being handled by a module.

23.12 TModuleItem

23.12.1 Description

`TModuleItem` holds registration information for an HTTP module in the module factory.

23.12.2 Interfaces overview

Page	Interfaces	Description
1234	IRouteInterface	Interface for route request handlers

23.12.3 Method overview

Page	Method	Description
791	Destroy	Destroys the module item

23.12.4 Property overview

Page	Properties	Access	Description
791	ModuleClass	rw	HTTP module class
791	ModuleName	rw	Name of the module
791	SkipStreaming	rw	Whether to skip component streaming

23.12.5 TModuleItem.Destroy

Synopsis: Destroys the module item

Declaration: `destructor Destroy;  Override`

Visibility: public

Description: `Destroy` frees the module item and its associated resources.

23.12.6 TModuleItem.ModuleClass

Synopsis: HTTP module class

Declaration: `Property ModuleClass : TCustomHTTPModuleClass`

Visibility: public

Access: Read,Write

Description: `ModuleClass` specifies the class for this registered HTTP module.

23.12.7 TModuleItem.ModuleName

Synopsis: Name of the module

Declaration: `Property ModuleName : string`

Visibility: public

Access: Read,Write

Description: `ModuleName` specifies the registered name for this HTTP module.

23.12.8 TModuleItem.SkipStreaming

Synopsis: Whether to skip component streaming

Declaration: `Property SkipStreaming : Boolean`

Visibility: public

Access: Read,Write

Description: `SkipStreaming` determines whether component streaming is skipped for this module.

23.13 TSessionFactory

23.13.1 Description

`TSessionFactory` provides session management functionality including creation, cleanup, and configuration.

23.13.2 Method overview

Page	Method	Description
792	CleanupSessions	Cleans up expired sessions
792	CreateSession	Creates a new user session
792	DoneSession	Finalizes a user session

23.13.3 Property overview

Page	Properties	Access	Description
792	CleanupInterval	rw	Session cleanup interval in minutes
793	DefaultTimeOutMinutes	rw	Default session timeout in minutes
793	SessionCookie	rw	Name of the session cookie
793	SessionCookiePath	rw	Path for the session cookie

23.13.4 TSessionFactory.CreateSession

Synopsis: Creates a new user session

Declaration: `function CreateSession(ARequest: TRequest) : TCustomSession`

Visibility: public

Description: `CreateSession` creates a new user session for the specified HTTP request.

23.13.5 TSessionFactory.DoneSession

Synopsis: Finalizes a user session

Declaration: `procedure DoneSession(var ASession: TCustomSession)`

Visibility: public

Description: `DoneSession` performs cleanup and finalization for the specified session.

23.13.6 TSessionFactory.CleanupSessions

Synopsis: Cleans up expired sessions

Declaration: `procedure CleanupSessions`

Visibility: public

Description: `CleanupSessions` removes expired sessions from the session storage.

23.13.7 TSessionFactory.CleanupInterval

Synopsis: Session cleanup interval in minutes

Declaration: `Property CleanupInterval : Integer`

Visibility: public

Access: Read,Write

Description: `CleanupInterval` specifies how often expired sessions are cleaned up, in minutes.

23.13.8 TSessionFactory.DefaultTimeOutMinutes

Synopsis: Default session timeout in minutes

Declaration: `Property DefaultTimeOutMinutes : Integer`

Visibility: public

Access: Read,Write

Description: `DefaultTimeOutMinutes` specifies the default timeout period for new sessions, in minutes.

23.13.9 TSessionFactory.SessionCookie

Synopsis: Name of the session cookie

Declaration: `Property SessionCookie : string`

Visibility: public

Access: Read,Write

Description: `SessionCookie` specifies the name of the HTTP cookie used to store session identifiers.

23.13.10 TSessionFactory.SessionCookiePath

Synopsis: Path for the session cookie

Declaration: `Property SessionCookiePath : string`

Visibility: public

Access: Read,Write

Description: `SessionCookiePath` specifies the path attribute for the session cookie.

23.14 TSessionHTTPModule

23.14.1 Description

`TSessionHTTPModule` extends HTTP modules with session management capabilities. It automatically manages user sessions across multiple HTTP requests.

23.14.2 Method overview

Page	Method	Description
794	destroy	Destroys the session HTTP module
794	HandleRequest	Handles request with session management

23.14.3 Property overview

Page	Properties	Access	Description
794	CreateSession	rw	Indicates if sessions should be created
794	OnNewSession	rw	Event triggered when a new session is created
795	OnSessionExpired	rw	Event triggered when a session expires
794	Session	rw	Current session object

23.14.4 TSessionHTTPModule.destroy

Synopsis: Destroys the session HTTP module

Declaration:
```
destructor destroy;  Override
```

Visibility: public

Description: `destroy` frees the session HTTP module and its associated session resources.

23.14.5 TSessionHTTPModule.HandleRequest

Synopsis: Handles request with session management

Declaration:
```
procedure HandleRequest(ARequest: TRequest; AResponse: TResponse)
                              ;  Override
```

Visibility: public

Description: `HandleRequest` processes an HTTP request with automatic session management.

23.14.6 TSessionHTTPModule.CreateSession

Synopsis: Indicates if sessions should be created

Declaration:
```
Property CreateSession : Boolean
```

Visibility: public

Access: Read,Write

Description: `CreateSession` determines whether new sessions are automatically created for requests.

23.14.7 TSessionHTTPModule.Session

Synopsis: Current session object

Declaration:
```
Property Session : TCustomSession
```

Visibility: public

Access: Read,Write

Description: `Session` provides access to the current user session associated with the request.

23.14.8 TSessionHTTPModule.OnNewSession

Synopsis: Event triggered when a new session is created

Declaration:
```
Property OnNewSession : TNotifyEvent
```

Visibility: public

Access: Read,Write

Description: `OnNewSession` is triggered when a new user session is created.

23.14.9 TSessionHTTPModule.OnSessionExpired

Synopsis: Event triggered when a session expires

Declaration: `Property OnSessionExpired : TNotifyEvent`

Visibility: public

Access: Read,Write

Description: `OnSessionExpired` is triggered when a user session expires due to timeout.

Chapter 24

Reference for unit 'fphttpclient'

24.1 Used units

Table 24.1: Used units by unit 'fphttpclient'

Name	Page
base64	200
Classes	??
HTTPDefs	1141
sslsockets	??
ssockets	??
System	??
sysutils	??
URIParser	1610

24.2 Overview

The `fpHttpClient` unit provides a complete HTTP client implementation for making HTTP requests. It supports all standard HTTP methods (GET, POST, PUT, DELETE, PATCH, OPTIONS, HEAD), SSL/TLS connections, redirects, authentication, cookies, and form posting including file uploads.

The main classes are TFPCustomHTTPClient (806) (base class) and TFPHTTPClient (816) (with published properties). The unit also supports Server-Sent Events through TCustomHTTPEventSource (796) and proxy connections through TProxyData (824).

24.3 Advanced TFPHTTPClient Features

The `TFPHTTPClient` class supports advanced features like authentication, custom headers, cookies, proxy settings, and request/response event handling. These features enable building robust HTTP client applications with full control over the HTTP communication.

```
program AdvancedHTTPExample;
```

```
{$mode objfpc}{$H+}

uses
  SysUtils, Classes, fphttpclient, opensslsockets, base64;

procedure OnDataReceived(Sender: TObject; const ContentLength, CurrentPos: Int64);
begin
  if ContentLength > 0 then
    Write(Format('Progress: %d/%d bytes (%.1f%%)' + #13,
                [CurrentPos, ContentLength, (CurrentPos/ContentLength)*100]));
end;

procedure OnPassword(Sender: TObject; var RepeatRequest: Boolean);
begin
  Writeln('Authentication required - this is where you would prompt for credentials'
  RepeatRequest := False; // Don't retry automatically
end;

var
  Client: TFPHTTPClient;
  Response: String;
  AuthHeader: String;
begin
  Client := TFPHTTPClient.Create(nil);
  try
    Writeln('=== Basic Authentication Example ===');
    // HTTP Basic Authentication
    AuthHeader := 'Basic ' + EncodeBase64('username:password');
    Client.AddHeader('Authorization', AuthHeader);

    // You can also use the Username/Password properties
    // Client.UserName := 'testuser';
    // Client.Password := 'testpass';

    Writeln('=== Custom Headers Example ===');
    Client.AddHeader('X-API-Key', 'your-api-key-here');
    Client.AddHeader('X-Custom-Header', 'custom-value');
    Client.AddHeader('Accept', 'application/json');

    Writeln('=== Event Handlers Example ===');
    Client.OnDataReceived := @OnDataReceived;
    Client.OnPassword := @OnPassword;

    Writeln('=== Cookie Management Example ===');
    // Cookies are automatically managed between requests
    Client.Cookies.Values['session_id'] := 'abc123';
    Client.Cookies.Values['user_pref'] := 'dark_mode';

    Writeln('=== Redirect Configuration ===');
    Client.AllowRedirect := True;
    Client.MaxRedirects := 10;

    Writeln('=== Timeout Configuration ===');
    Client.IOTimeout := 30000; // 30 seconds timeout
```

```
    Client.ConnectTimeout := 10000; // 10 seconds connect timeout

    Writeln('=== Proxy Configuration Example ===');
    // Uncomment to use proxy
    // Client.Proxy.Host := 'proxy.example.com';
    // Client.Proxy.Port := 8080;
    // Client.Proxy.UserName := 'proxy_user';
    // Client.Proxy.Password := 'proxy_pass';

    try
      // Make request with all configured settings
      Response := Client.Get('https://httpbin.org/headers');

      Writeln('Request completed successfully!');
      Writeln('Status Code: ', Client.ResponseStatusCode);
      Writeln('Response Length: ', Length(Response));

      // Access response headers
      Writeln('Server: ', Client.GetHeader(Client.ResponseHeaders, 'Server'));
      Writeln('Content-Type: ', Client.GetHeader(Client.ResponseHeaders, 'Content-Ty

      // Show cookies received from server
      if Client.Cookies.Count > 0 then
      begin
        Writeln('Cookies received:');
        Writeln(Client.Cookies.Text);
      end;

    except
      on E: Exception do
      begin
        Writeln('Request failed: ', E.Message);
        Writeln('Last status code: ', Client.ResponseStatusCode);
      end;
    end;

  finally
    Client.Free;
  end;
end.
```

24.4 File Upload with Local Filename

The `FileFormPost` method provides an easy way to upload local files to a server. It automatically handles multipart/form-data encoding and sets appropriate headers. The method takes the URL, form field name, local filename, and a response stream.

```
program FileUploadExample;

{$mode objfpc}{$H+}

uses
```

```
  SysUtils, Classes, fphttpclient, opensslsockets;

procedure CreateTestFile(const FileName: String);
var
  F: TextFile;
begin
  AssignFile(F, FileName);
  Rewrite(F);
  try
    Writeln(F, 'Sample file content for upload test');
    Writeln(F, 'Created: ', DateTimeToStr(Now));
    Writeln(F, 'This file will be uploaded via HTTP POST');
  finally
    CloseFile(F);
  end;
end;

var
  Client: TFPHTTPClient;
  ResponseStream: TStringStream;
  TestFileName: String;
begin
  Client := TFPHTTPClient.Create(nil);
  ResponseStream := TStringStream.Create('');
  TestFileName := '/tmp/claude/upload_test.txt';

  try
    // Create a test file to upload
    CreateTestFile(TestFileName);

    Writeln('=== File Upload with Local Filename ===');
    Writeln('Uploading file: ', TestFileName);

    // Configure client for file upload
    Client.AllowRedirect := True;
    Client.AddHeader('User-Agent', 'FreePascal-FileUploader/1.0');

    // Upload the file
    // Parameters: URL, field name, local filename, response stream
    Client.FileFormPost('https://httpbin.org/post', 'uploaded_file', TestFileName, R

    // Check upload result
    if Client.ResponseStatusCode = 200 then
    begin
      Writeln('Upload successful!');
      Writeln('Server response status: ', Client.ResponseStatusCode);
      Writeln('Response content type: ', Client.GetHeader(Client.ResponseHeaders, 'C

      // Show part of the response
      Writeln('Server response (first 200 chars):');
      Writeln(Copy(ResponseStream.DataString, 1, 200));
    end
    else
    begin
```

```
      Writeln('Upload failed with status: ', Client.ResponseStatusCode);
      Writeln('Response: ', ResponseStream.DataString);
    end;

    // Upload with additional form fields
    Writeln('=== File Upload with Additional Fields ===');
    ResponseStream.Clear;

    // Add custom headers for the upload
    Client.AddHeader('X-Upload-Purpose', 'documentation-example');

    // You can also upload to services that expect additional form data
    // by using FileFormPost combined with other form fields

  finally
    ResponseStream.Free;
    Client.Free;

    // Clean up test file
    if FileExists(TestFileName) then
      DeleteFile(TestFileName);
  end;
end.
```

Important Notes:

- The file must exist and be readable for the upload to succeed.
- The form field name should match what the server expects.
- The method automatically sets Content-Type to multipart/form-data
- Large files are streamed efficiently without loading entirely into memory
- The server response is written to the provided response stream

24.5 Form Data and File Upload with fpHttpClient

The `TFPHTTPClient` class provides specialized methods for posting form data and uploading files. `FormPost` handles URL-encoded form data, while `FileFormPost` uploads files using multipart/form-data encoding.

```
program FormPostExample;

{$mode objfpc}{$H+}

uses
  SysUtils, Classes, fphttpclient, opensslsockets;

var
  Client: TFPHTTPClient;
  FormData: TStrings;
  Response: String;
```

```
  FileStream: TFileStream;
begin
  Client := TFPHTTPClient.Create(nil);
  FormData := TStringList.Create;
  try
    // Simple form post with key-value pairs
    Writeln('=== Form Post Example ===');
    FormData.Values['username'] := 'testuser';
    FormData.Values['email'] := 'test@example.com';
    FormData.Values['message'] := 'Hello from FreePascal!';

    Response := Client.FormPost('https://httpbin.org/post', FormData);
    Writeln('Form POST response: ', Copy(Response, 1, 150), '...');

    // Form post with raw data
    Writeln('=== Raw Form Data Post ===');
    Response := Client.FormPost('https://httpbin.org/post', 'field1=value1&field2=va
    Writeln('Raw form POST status: ', Client.ResponseStatusCode);

    // File upload using FileFormPost
    Writeln('=== File Upload Example ===');

    // Create a test file to upload
    FileStream := TFileStream.Create('/tmp/claude/test_upload.txt', fmCreate);
    try
      FileStream.WriteAnsiString('This is test content for file upload');
    finally
      FileStream.Free;
    end;

    // Upload the file
    Client.FileFormPost('https://httpbin.org/post', 'file', '/tmp/claude/test_upload
                        TStringStream.Create(''));

    Writeln('File upload status: ', Client.ResponseStatusCode);
    if Client.ResponseStatusCode = 200 then
      Writeln('File uploaded successfully');

    // Clean up test file
    if FileExists('/tmp/claude/test_upload.txt') then
      DeleteFile('/tmp/claude/test_upload.txt');

  finally
    FormData.Free;
    Client.Free;
  end;
end.
```

24.6 HTTP Methods with TFPHTTPClient

The `TFPHTTPClient` class supports all standard HTTP methods through various methods and overloads. Each method provides multiple ways to handle the response: as a string, stream, or file.

```
program HTTPMethodsExample;

{$mode objfpc}{$H+}

uses
  SysUtils, Classes, fphttpclient, opensslsockets;

var
  Client: TFPHTTPClient;
  Response: String;
  PostData: TStringStream;
  ResponseStream: TFileStream;
begin
  Client := TFPHTTPClient.Create(nil);
  try
    // GET request
    Writeln('=== GET Request ===');
    Response := Client.Get('https://httpbin.org/get?param1=value1');
    Writeln('GET Response: ', Copy(Response, 1, 100), '...');

    // POST request with data
    Writeln('=== POST Request ===');
    Client.RequestBody := TRawByteStringStream.Create('{"key": "value", "number": 42
    Client.AddHeader('Content-Type', 'application/json');
    Response := Client.Post('https://httpbin.org/post');
    Writeln('POST Response: ', Copy(Response, 1, 100), '...');

    // PUT request
    Writeln('=== PUT Request ===');
    Client.RequestBody := TRawByteStringStream.Create('{"updated": "data"}');
    Client.AddHeader('Content-Type', 'application/json');
    Client.HTTPMethod('PUT', 'https://httpbin.org/put', nil, [200]);
    Writeln('PUT Status Code: ', Client.ResponseStatusCode);

    // DELETE request
    Writeln('=== DELETE Request ===');
    Client.HTTPMethod('DELETE', 'https://httpbin.org/delete', nil, [200]);
    Writeln('DELETE Status Code: ', Client.ResponseStatusCode);

    // HEAD request (only headers, no body)
    Writeln('=== HEAD Request ===');
    Client.HTTPMethod('HEAD', 'https://httpbin.org/get', nil, [200]);
    Writeln('HEAD Content-Length: ', Client.GetHeader(Client.ResponseHeaders, 'Conte

    // OPTIONS request
    Writeln('=== OPTIONS Request ===');
    Client.HTTPMethod('OPTIONS', 'https://httpbin.org', nil, [200]);
    Writeln('Allowed Methods: ', Client.GetHeader(Client.ResponseHeaders, 'Allow'));

    // Save response to file
    Writeln('=== GET to File ===');
    Client.Get('https://httpbin.org/json', '/tmp/claude/response.json');
    if FileExists('/tmp/claude/response.json') then
      Writeln('Response saved to file successfully');
```

```
  finally
    Client.Free;
  end;
end.
```

24.7 HTTPS Support in TFPHTTPClient

To enable HTTPS support in the TFPHTTPClient class, you must include either `opensslsockets` or `gnutlssockets` in your program's uses clause. These units register SSL socket handlers that the HTTP client uses automatically when encountering HTTPS URLs.

`opensslsockets` uses the OpenSSL library and is the most commonly used option. `gnutlssockets` uses the GnuTLS library as an alternative. Include only one of these units - do not include both in the same program.

```
program HTTPSExample;

{$mode objfpc}{$H+}

uses
  SysUtils, fphttpclient, opensslsockets; // This enables HTTPS support

var
  Client: TFPHTTPClient;
  Response: String;
begin
  Client := TFPHTTPClient.Create(nil);
  try
    // Configure SSL settings if needed
    Client.AllowRedirect := True;

    // Make HTTPS request - automatically uses SSL
    Response := Client.Get('https://api.github.com/users/freepascal');

    Writeln('HTTPS Response:');
    Writeln(Response);

    Writeln('HTTPS request completed successfully');
    Writeln('Status Code: ', Client.ResponseStatusCode);

  finally
    Client.Free;
  end;
end.
```

SSL Certificate Verification: By default, the client verifies SSL certificates. You can disable this verification for testing purposes by setting `Client.IOTimeout := 0` and handling the `OnVerifyCertificate` event, but this is not recommended for production code as it makes connections vulnerable to man-in-the-middle attacks.

24.8 Using the TFPHTTPClient

The `TFPHTTPClient` class provides both simple class methods for quick requests and instance-based methods for more control over the HTTP client behavior. For basic operations, use the static `SimpleGet`, `SimplePost`, and `SimpleFormPost` methods. For advanced features like custom headers, authentication, or cookie management, create a `TFPHTTPClient` instance.

The HTTP client automatically handles redirects (up to a configurable limit), manages cookies, and supports both HTTP and HTTPS protocols. For HTTPS support, you must include either `opensslsockets` or `gnutlssockets` in your program's uses clause.

```
program HTTPClientBasic;

{$mode objfpc}{$H+}

uses
  SysUtils, Classes, fphttpclient, opensslsockets; // opensslsockets enables HTTPS

var
  Client: TFPHTTPClient;
  Response: String;
  ResponseStream: TStringStream;
begin
  // Simple GET request using class method
  Response := TFPHTTPClient.SimpleGet('https://httpbin.org/get');
  Writeln('Simple GET response:');
  Writeln(Response);

  // Instance-based approach for more control
  Client := TFPHTTPClient.Create(nil);
  ResponseStream := TStringStream.Create('');
  try
    // Configure client
    Client.AllowRedirect := True;
    Client.MaxRedirects := 5;

    // Add custom headers
    Client.AddHeader('User-Agent', 'MyApp/1.0');
    Client.AddHeader('Accept', 'application/json');

    // Make request to stream
    Client.Get('https://httpbin.org/headers', ResponseStream);

    Writeln('Response with custom headers:');
    Writeln(ResponseStream.DataString);

    Writeln('Response Status Code: ', Client.ResponseStatusCode);
    Writeln('Response Headers:');
    Writeln(Client.ResponseHeaders.Text);
  finally
    ResponseStream.Free;
    Client.Free;
  end;
end.
```

24.9 Constants, types and variables

24.9.1 Constants

```
DefMaxRedirects = 16
```

Default maximum number of HTTP redirects that will be followed automatically. The value is 16, which prevents infinite redirect loops while allowing reasonable redirect chains.

```
ReadBufLen = 4096
```

Default buffer size used for reading data from sockets. The value is 4096 bytes, which provides a good balance between memory usage and performance.

24.9.2 Types

```
TDataEvent = procedure(Sender: TObject; const ContentLength: Int64
  ;
                       const CurrentPos: Int64) of object
```

Event type called during data transfer to provide progress information. During header reading, ContentLength is 0. During content reading, ContentLength is -1 if not specified by server.

```
TGetSocketHandlerEvent = procedure(Sender: TObject;
  const UseSSL: Boolean;
  out AHandler: TSocketHandler)
  of object
```

Event type called to allow custom socket handler creation. This enables the use of custom SSL implementations or socket configurations.

```
THTTPVerifyCertificateEvent = procedure(Sender: TObject;
  AHandler: TSSLSocketHandler;
  var aAllow: Boolean) of
  object
```

Event type called during SSL certificate verification, allowing custom certificate validation logic.

```
TPasswordEvent = procedure(Sender: TObject; var RepeatRequest: Boolean
  )
                            of object
```

Event type called when the server requests authentication. The event handler can provide credentials and indicate whether to retry the request.

```
TRedirectEvent = procedure(Sender: TObject; const ASrc: string;
  var ADest: string) of object
```

Event type called when an HTTP redirect is encountered. The event handler can modify the destination URL or prevent the redirect.

```
TSocketHandlerCreatedEvent = procedure(Sender: TObject;
  AHandler: TSocketHandler)
  of object
```

Event type called after a socket handler has been created, allowing additional configuration of the handler.

24.10 Procedures and functions

24.10.1 DecodeURLElement

Synopsis: URL-decodes a string from URL encoding

Declaration: `function DecodeURLElement(const S: string) : string`

Visibility: default

Description: Decodes percent-encoded sequences in a URL-encoded string back to their original characters.

See also: EncodeURLElement (806)

24.10.2 EncodeURLElement

Synopsis: URL-encodes a string for use in URLs

Declaration: `function EncodeURLElement(S: string) : string`

Visibility: default

Description: Encodes special characters in a string to make it safe for use in URLs by replacing them with percent-encoded sequences.

See also: DecodeURLElement (806)

24.11 EHTTPClient

24.11.1 Description

`EHTTPClient` is raised when HTTP operations fail, providing status code and error message information.

See also: EHTTPClientSocket (796)

24.12 TFPCustomHTTPClient

24.12.1 Description

`TFPCustomHTTPClient` is the base class for HTTP client functionality. It provides methods for all standard HTTP operations including GET, POST, PUT, DELETE, PATCH, OPTIONS, and HEAD requests.

The class supports SSL/TLS connections, HTTP redirects, authentication, cookies, form posting with file uploads, proxy connections, and Server-Sent Events. It handles connection management, timeout control, and provides various events for monitoring request progress.

See also: TFPHTTPClient (816), TProxyData (824)

24.12.2 Method overview

Page	Method	Description
808	AddHeader	Adds an HTTP header to a headers collection
807	Create	Creates a new HTTP client instance
811	Delete	Performs an HTTP DELETE request
807	Destroy	Destroys the HTTP client instance
815	FileFormPost	Performs an HTTP POST request with file upload
814	FormPost	Performs an HTTP POST request with form data
809	Get	Performs an HTTP GET request
808	GetHeader	Gets the value of a header from a headers collection
813	Head	Performs an HTTP HEAD request
809	HTTPMethod	Executes an HTTP request with the specified method
808	IndexOfHeader	Returns the index of a header in a headers collection
809	IsRedirect	Checks if an HTTP status code indicates a redirect
813	Options	Performs an HTTP OPTIONS request
812	Patch	Performs an HTTP PATCH request
810	Post	Performs an HTTP POST request
811	Put	Performs an HTTP PUT request
810	RedirectForcesGET	Checks if a redirect status code forces GET method
812	SimpleDelete	Static method to perform a simple HTTP DELETE request
815	SimpleFileFormPost	Static method to perform a simple file upload form POST
814	SimpleFormPost	Static method to perform a simple form POST request
810	SimpleGet	Static method to perform a simple HTTP GET request
813	SimpleOptions	Static method to perform a simple HTTP OPTIONS request
812	SimplePatch	Static method to perform a simple HTTP PATCH request
810	SimplePost	Static method to perform a simple HTTP POST request
811	SimplePut	Static method to perform a simple HTTP PUT request
815	StreamFormPost	Performs an HTTP form POST with stream upload
809	Terminate	Terminates any ongoing HTTP operation

24.12.3 Property overview

Page	Properties	Access	Description
816	Terminated	r	Indicates whether the HTTP operation has been terminated

24.12.4 TFPCustomHTTPClient.Create

Synopsis: Creates a new HTTP client instance

Declaration: `constructor Create(AOwner: TComponent); Override`

Visibility: public

Description: Creates a new HTTP client instance and initializes default settings for timeouts, HTTP version, and other configuration options.

See also: Destroy (807)

24.12.5 TFPCustomHTTPClient.Destroy

Synopsis: Destroys the HTTP client instance

Declaration: `destructor Destroy; Override`

Visibility: public

Description: Destroys the HTTP client instance, closes any open connections, and frees associated resources.

See also: Create (807)

24.12.6 TFPCustomHTTPClient.AddHeader

Synopsis: Adds an HTTP header to a headers collection

Declaration:
```
class procedure AddHeader(HTTPHeaders: TStrings; const AHeader: string;
                          const AValue: string)
procedure AddHeader(const AHeader: string; const AValue: string)
```

Visibility: public

Description: Static utility method to add an HTTP header with the specified name and value to a headers collection. If a header with the same name already exists, its value is updated.

See also: GetHeader (808)

24.12.7 TFPCustomHTTPClient.IndexOfHeader

Synopsis: Returns the index of a header in a headers collection

Declaration:
```
class function IndexOfHeader(HTTPHeaders: TStrings;
                             const AHeader: string) : Integer
function IndexOfHeader(const AHeader: string) : Integer
```

Visibility: public

Description: Static utility method to find the index of a header with the specified name in a headers collection. Returns -1 if the header is not found.

See also: GetHeader (808)

24.12.8 TFPCustomHTTPClient.GetHeader

Synopsis: Gets the value of a header from a headers collection

Declaration:
```
class function GetHeader(HTTPHeaders: TStrings; const AHeader: string)
                         : string
function GetHeader(const AHeader: string) : string
```

Visibility: public

Description: Static utility method to retrieve the value of a header with the specified name from a headers collection. Returns an empty string if the header is not found.

See also: AddHeader (808)

24.12.9 TFPCustomHTTPClient.Terminate

Synopsis: Terminates any ongoing HTTP operation

Declaration:
```
procedure Terminate
```

Visibility: public

Description: Signals that any ongoing HTTP operation should be terminated. This sets the `Terminated` property to `True` and may abort current requests.

See also: Terminated (816)

24.12.10 TFPCustomHTTPClient.HTTPMethod

Synopsis: Executes an HTTP request with the specified method

Declaration:
```
procedure HTTPMethod(const AMethod: string; const AURL: string;
                     Stream: TStream;
                     const AllowedResponseCodes: Array of Integer)
                     ;  Virtual
```

Visibility: public

Description: Generic method for executing HTTP requests with any HTTP method. This is the core method used by all other HTTP method shortcuts (GET, POST, etc.).

Errors: May raise HTTP-related exceptions on connection errors or invalid responses.

See also: Get (809), Post (810)

24.12.11 TFPCustomHTTPClient.Get

Synopsis: Performs an HTTP GET request

Declaration:
```
procedure Get(const AURL: string; Stream: TStream)
procedure Get(const AURL: string; const LocalFileName: string)
procedure Get(const AURL: string; Response: TStrings)
function Get(const AURL: string) : RawByteString
```

Visibility: public

Description: Performs an HTTP GET request to retrieve data from the specified URL. The response can be written to a stream, saved to a file, or returned as a string.

Errors: May raise HTTP-related exceptions on connection errors or invalid responses.

See also: Post (810), SimpleGet (810)

24.12.12 TFPCustomHTTPClient.IsRedirect

Synopsis: Checks if an HTTP status code indicates a redirect

Declaration:
```
class function IsRedirect(ACode: Integer) : Boolean;  Virtual
```

Visibility: public

Description: Static utility method that returns `True` if the specified HTTP status code indicates a redirect (3xx codes).

See also: RedirectForcesGET (810)

24.12.13 TFPCustomHTTPClient.RedirectForcesGET

Synopsis: Checks if a redirect status code forces GET method

Declaration:
```
class function RedirectForcesGET(ACode: Integer) : Boolean;  Virtual
```

Visibility: public

Description: Static utility method that returns `True` if the specified redirect status code requires changing the HTTP method to GET. This applies to certain 3xx status codes that mandate method changes.

See also: IsRedirect (809)

24.12.14 TFPCustomHTTPClient.SimpleGet

Synopsis: Static method to perform a simple HTTP GET request

Declaration:
```
class procedure SimpleGet(const AURL: string; Stream: TStream)
class procedure SimpleGet(const AURL: string;
                          const LocalFileName: string)
class procedure SimpleGet(const AURL: string; Response: TStrings)
class function SimpleGet(const AURL: string) : RawByteString
```

Visibility: public

Description: Static utility method for performing a simple HTTP GET request without requiring an HTTP client instance. This is convenient for quick one-off requests.

Errors: May raise HTTP-related exceptions on connection errors.

See also: Get (809)

24.12.15 TFPCustomHTTPClient.Post

Synopsis: Performs an HTTP POST request

Declaration:
```
procedure Post(const URL: string; const Response: TStream)
procedure Post(const URL: string; Response: TStrings)
procedure Post(const URL: string; const LocalFileName: string)
function Post(const URL: string) : RawByteString
```

Visibility: public

Description: Performs an HTTP POST request to send data to the specified URL. The request body is taken from the `RequestBody` property.

Errors: May raise HTTP-related exceptions on connection errors or invalid responses.

See also: Get (809), FormPost (814)

24.12.16 TFPCustomHTTPClient.SimplePost

Synopsis: Static method to perform a simple HTTP POST request

Declaration:
```
class procedure SimplePost(const URL: string; const Response: TStream)
class procedure SimplePost(const URL: string; Response: TStrings)
class procedure SimplePost(const URL: string;
                           const LocalFileName: string)
class function SimplePost(const URL: string) : RawByteString
```

Visibility: public

Description: Static utility method for performing a simple HTTP POST request without requiring an HTTP client instance.

Errors: May raise HTTP-related exceptions on connection errors.

See also: Post (810)

24.12.17 TFPCustomHTTPClient.Put

Synopsis: Performs an HTTP PUT request

Declaration:
```
procedure Put(const URL: string; const Response: TStream)
procedure Put(const URL: string; Response: TStrings)
procedure Put(const URL: string; const LocalFileName: string)
function Put(const URL: string) : RawByteString
```

Visibility: public

Description: Performs an HTTP PUT request to upload or update data at the specified URL. The request body is taken from the `RequestBody` property.

Errors: May raise HTTP-related exceptions on connection errors or invalid responses.

See also: Post (810), Patch (812)

24.12.18 TFPCustomHTTPClient.SimplePut

Synopsis: Static method to perform a simple HTTP PUT request

Declaration:
```
class procedure SimplePut(const URL: string; const Response: TStream)
class procedure SimplePut(const URL: string; Response: TStrings)
class procedure SimplePut(const URL: string;
                          const LocalFileName: string)
class function SimplePut(const URL: string) : RawByteString
```

Visibility: public

Description: Static utility method for performing a simple HTTP PUT request without requiring an HTTP client instance.

Errors: May raise HTTP-related exceptions on connection errors.

See also: Put (811)

24.12.19 TFPCustomHTTPClient.Delete

Synopsis: Performs an HTTP DELETE request

Declaration:
```
procedure Delete(const URL: string; const Response: TStream)
procedure Delete(const URL: string; Response: TStrings)
procedure Delete(const URL: string; const LocalFileName: string)
function Delete(const URL: string) : RawByteString
```

Visibility: public

Description: Performs an HTTP DELETE request to delete a resource at the specified URL.

Errors: May raise HTTP-related exceptions on connection errors or invalid responses.

See also: Get (809), Post (810)

24.12.20 TFPCustomHTTPClient.SimpleDelete

Synopsis: Static method to perform a simple HTTP DELETE request

Declaration:
```
class procedure SimpleDelete(const URL: string; const Response: TStream)
class procedure SimpleDelete(const URL: string; Response: TStrings)
class procedure SimpleDelete(const URL: string;
                             const LocalFileName: string)
class function SimpleDelete(const URL: string) : RawByteString
```

Visibility: public

Description: Static utility method for performing a simple HTTP DELETE request without requiring an HTTP client instance.

Errors: May raise HTTP-related exceptions on connection errors.

See also: Delete (811)

24.12.21 TFPCustomHTTPClient.Patch

Synopsis: Performs an HTTP PATCH request

Declaration:
```
procedure Patch(const URL: string; const Response: TStream)
procedure Patch(const URL: string; Response: TStrings)
procedure Patch(const URL: string; const LocalFileName: string)
function Patch(const URL: string) : RawByteString
```

Visibility: public

Description: Performs an HTTP PATCH request to partially update a resource at the specified URL. The request body is taken from the `RequestBody` property.

Errors: May raise HTTP-related exceptions on connection errors or invalid responses.

See also: Put (811), Post (810)

24.12.22 TFPCustomHTTPClient.SimplePatch

Synopsis: Static method to perform a simple HTTP PATCH request

Declaration:
```
class procedure SimplePatch(const URL: string; const Response: TStream)
class procedure SimplePatch(const URL: string; Response: TStrings)
class procedure SimplePatch(const URL: string;
                            const LocalFileName: string)
class function SimplePatch(const URL: string) : RawByteString
```

Visibility: public

Description: Static utility method for performing a simple HTTP PATCH request without requiring an HTTP client instance.

Errors: May raise HTTP-related exceptions on connection errors.

See also: Patch (812)

24.12.23 TFPCustomHTTPClient.Options

Synopsis: Performs an HTTP OPTIONS request

Declaration:
```
procedure Options(const URL: string; const Response: TStream)
procedure Options(const URL: string; Response: TStrings)
procedure Options(const URL: string; const LocalFileName: string)
function Options(const URL: string) : RawByteString
```

Visibility: public

Description: Performs an HTTP OPTIONS request to query the supported methods and capabilities of a server or resource.

Errors: May raise HTTP-related exceptions on connection errors or invalid responses.

See also: Head (813)

24.12.24 TFPCustomHTTPClient.SimpleOptions

Synopsis: Static method to perform a simple HTTP OPTIONS request

Declaration:
```
class procedure SimpleOptions(const URL: string;
                              const Response: TStream)
class procedure SimpleOptions(const URL: string; Response: TStrings)
class procedure SimpleOptions(const URL: string;
                              const LocalFileName: string)
class function SimpleOptions(const URL: string) : RawByteString
```

Visibility: public

Description: Static utility method for performing a simple HTTP OPTIONS request without requiring an HTTP client instance.

Errors: May raise HTTP-related exceptions on connection errors.

See also: Options (813)

24.12.25 TFPCustomHTTPClient.Head

Synopsis: Performs an HTTP HEAD request

Declaration:
```
class procedure Head(AURL: string; Headers: TStrings)
```

Visibility: public

Description: Performs an HTTP HEAD request to retrieve only the headers for a resource without the body content. Useful for checking resource metadata, existence, or modification dates.

Errors: May raise HTTP-related exceptions on connection errors or invalid responses.

See also: Get (809)

24.12.26 TFPCustomHTTPClient.FormPost

Synopsis: Performs an HTTP POST request with form data

Declaration:
```
procedure FormPost(const URL: string; FormData: RawByteString;
                   const Response: TStream)
procedure FormPost(const URL: string; FormData: TStrings;
                   const Response: TStream)
procedure FormPost(const URL: string; const FormData: string;
                   const Response: TStrings)
procedure FormPost(const URL: string; FormData: TStrings;
                   const Response: TStrings)
function FormPost(const URL: string; const FormData: RawByteString)
                   : RawByteString
function FormPost(const URL: string; FormData: TStrings) : RawByteString
```

Visibility: public

Description: Performs an HTTP POST request with form-encoded data. The form data is automatically encoded as application/x-www-form-urlencoded.

Errors: May raise HTTP-related exceptions on connection errors or invalid responses.

See also: Post (810), FileFormPost (815)

24.12.27 TFPCustomHTTPClient.SimpleFormPost

Synopsis: Static method to perform a simple form POST request

Declaration:
```
class procedure SimpleFormPost(const URL: string;
                               const FormData: RawByteString;
                               const Response: TStream)
class procedure SimpleFormPost(const URL: string; FormData: TStrings;
                               const Response: TStream)
class procedure SimpleFormPost(const URL: string;
                               const FormData: RawByteString;
                               const Response: TStrings)
class procedure SimpleFormPost(const URL: string; FormData: TStrings;
                               const Response: TStrings)
class function SimpleFormPost(const URL: string;
                              const FormData: RawByteString)
                               : RawByteString
class function SimpleFormPost(const URL: string; FormData: TStrings)
                               : RawByteString
```

Visibility: public

Description: Static utility method for performing a simple form POST request without requiring an HTTP client instance.

Errors: May raise HTTP-related exceptions on connection errors.

See also: FormPost (814)

24.12.28 TFPCustomHTTPClient.FileFormPost

Synopsis: Performs an HTTP POST request with file upload

Declaration:
```
procedure FileFormPost(const AURL: string; const AFieldName: string;
                       const AFileName: string; const Response: TStream)
procedure FileFormPost(const AURL: string; FormData: TStrings;
                       AFieldName: string; AFileName: string;
                       const Response: TStream)
```

Visibility: public

Description: Performs an HTTP POST request with file upload using multipart/form-data encoding. This method allows uploading files as part of a form submission.

Errors: May raise HTTP-related exceptions on connection errors or invalid responses.

See also: FormPost (814), StreamFormPost (815)

24.12.29 TFPCustomHTTPClient.StreamFormPost

Synopsis: Performs an HTTP form POST with stream upload

Declaration:
```
procedure StreamFormPost(const AURL: string; const AFieldName: string;
                         const AFileName: string;
                         const AStream: TStream; const Response: TStream)
procedure StreamFormPost(const AURL: string; FormData: TStrings;
                         const AFieldName: string;
                         const AFileName: string;
                         const AStream: TStream; const Response: TStream)
```

Visibility: public

Description: Performs an HTTP POST request with form data that includes a file uploaded from a stream. Uses multipart/form-data encoding for the upload.

Errors: May raise HTTP-related exceptions on connection errors or invalid responses.

See also: FileFormPost (815), FormPost (814)

24.12.30 TFPCustomHTTPClient.SimpleFileFormPost

Synopsis: Static method to perform a simple file upload form POST

Declaration:
```
class procedure SimpleFileFormPost(const AURL: string;
                                   const AFieldName: string;
                                   const AFileName: string;
                                   const Response: TStream)
```

Visibility: public

Description: Static utility method for performing a simple file upload form POST request without requiring an HTTP client instance. Uses multipart/form-data encoding for the file upload.

Errors: May raise HTTP-related exceptions on connection errors.

See also: FileFormPost (815)

24.12.31 TFPCustomHTTPClient.Terminated

Synopsis: Indicates whether the HTTP operation has been terminated

Declaration: `Property Terminated : Boolean`

Visibility: public

Access: Read

Description: Returns `True` if the current HTTP operation has been terminated by calling `Terminate`. This can be used to check if an operation was cancelled.

See also: Terminate (809)

24.13 TFPHTTPClient

24.13.1 Description

`TFPHTTPClient` is the complete HTTP client implementation that publishes all properties from TFPCustomHTTPClient (806). This class is intended for use in visual design environments and provides access to all HTTP client functionality through published properties.

See also: TFPCustomHTTPClient (806)

24.13.2 Property overview

Page	Properties	Access	Description
823	AfterSocketHandlerCreate		Event called after socket handler creation
820	AllowRedirect		Whether to automatically follow HTTP redirects
817	Connected		Indicates whether a connection is currently active
818	ConnectTimeout		Timeout for connection establishment in milliseconds
820	Cookies		Cookie jar for automatic cookie management
819	HTTPversion		The HTTP protocol version to use for requests
818	IOTimeout		Timeout for data transfer operations in milliseconds
817	KeepConnection		Whether to keep the connection alive between requests
821	MaxRedirects		Maximum number of redirects to follow automatically
822	OnDataReceived		Event called during data reception to track progress
822	OnGetSocketHandler		Event called to provide custom socket handlers
822	OnHeaders		Event called when response headers are received
822	OnPassword		Event called when authentication is required
821	OnRedirect		Event called when a redirect is encountered
823	OnVerifySSLCertificate		Event called for custom SSL certificate verification
821	Password		Password for HTTP authentication
823	Proxy		Proxy server configuration
819	RequestBody		The request body content for POST/PUT requests
818	RequestHeaders		HTTP headers to send with requests
819	ResponseHeaders		HTTP headers received from server responses
820	ResponseStatusCode		HTTP status code from the last response
820	ResponseStatusText		HTTP status text from the last response
819	ServerHTTPVersion		The HTTP protocol version used by the server in the last response
821	UserName		Username for HTTP authentication
823	VerifySSLCertificate		Whether to verify SSL certificates

24.13.3 TFPHTTPClient.KeepConnection

Synopsis: Whether to keep the connection alive between requests

Declaration: `Property KeepConnection :`

Visibility: published

Access:

Description: When set to `True`, the HTTP connection is kept open between requests to the same server. This can improve performance for multiple requests to the same host.

See also: Connected (817)

24.13.4 TFPHTTPClient.Connected

Synopsis: Indicates whether a connection is currently active

Declaration: `Property Connected :`

Visibility: published

Access:

Description: Returns `True` if there is currently an active connection to a server.

See also: KeepConnection (817)

24.13.5 TFPHTTPClient.IOTimeout

Synopsis: Timeout for data transfer operations in milliseconds

Declaration: `Property IOTimeout :`

Visibility: published

Access:

Description: Maximum time to wait for data transfer operations (reading/writing) before timing out. A value of 0 means no timeout.

See also: ConnectTimeout (818)

24.13.6 TFPHTTPClient.ConnectTimeout

Synopsis: Timeout for connection establishment in milliseconds

Declaration: `Property ConnectTimeout :`

Visibility: published

Access:

Description: Maximum time to wait when establishing a connection to a server before timing out. A value of 0 means no timeout.

See also: IOTimeout (818)

24.13.7 TFPHTTPClient.RequestHeaders

Synopsis: HTTP headers to send with requests

Declaration: `Property RequestHeaders :`

Visibility: published

Access:

Description: Collection of HTTP headers that will be sent with all requests. Common headers include Content-Type, User-Agent, Accept, etc.

See also: ResponseHeaders (819)

24.13.8 TFPHTTPClient.RequestBody

Synopsis: The request body content for POST/PUT requests

Declaration: `Property RequestBody :`

Visibility: published

Access:

Description: Contains the data that will be sent as the request body for POST, PUT, PATCH, and similar requests. The content type should be set appropriately in `RequestHeaders`.

See also: RequestHeaders (818)

24.13.9 TFPHTTPClient.ResponseHeaders

Synopsis: HTTP headers received from server responses

Declaration: `Property ResponseHeaders :`

Visibility: published

Access:

Description: Collection of HTTP headers received from the server in the last response. This is populated after each request and contains server response metadata.

See also: RequestHeaders (818), ResponseStatusCode (820)

24.13.10 TFPHTTPClient.HTTPversion

Synopsis: The HTTP protocol version to use for requests

Declaration: `Property HTTPversion :`

Visibility: published

Access:

Description: Specifies which HTTP protocol version to use (e.g., '1.0', '1.1'). Defaults to HTTP/1.1.

See also: ServerHTTPVersion (819)

24.13.11 TFPHTTPClient.ServerHTTPVersion

Synopsis: The HTTP protocol version used by the server in the last response

Declaration: `Property ServerHTTPVersion :`

Visibility: published

Access:

Description: Contains the HTTP protocol version that the server used in its last response.

See also: HTTPversion (819)

24.13.12 TFPHTTPClient.ResponseStatusCode

Synopsis: HTTP status code from the last response

Declaration: `Property ResponseStatusCode :`

Visibility: published

Access:

Description: The HTTP status code returned by the server for the last request. Common values include 200 (OK), 404 (Not Found), 500 (Internal Server Error), etc.

See also: ResponseStatusText (820), ResponseHeaders (819)

24.13.13 TFPHTTPClient.ResponseStatusText

Synopsis: HTTP status text from the last response

Declaration: `Property ResponseStatusText :`

Visibility: published

Access:

Description: The HTTP status text (reason phrase) returned by the server for the last request. For example, 'OK' for status code 200, 'Not Found' for 404, etc.

See also: ResponseStatusCode (820)

24.13.14 TFPHTTPClient.Cookies

Synopsis: Cookie jar for automatic cookie management

Declaration: `Property Cookies :`

Visibility: published

Access:

Description: Cookie jar that automatically manages cookies between requests. Cookies received from servers are stored and automatically sent with subsequent requests to the same domain.

See also: RequestCookies (816), ResponseCookies (816)

24.13.15 TFPHTTPClient.AllowRedirect

Synopsis: Whether to automatically follow HTTP redirects

Declaration: `Property AllowRedirect :`

Visibility: published

Access:

Description: When set to `True`, the client will automatically follow HTTP redirects (3xx status codes). When `False`, redirects must be handled manually.

See also: MaxRedirects (821), OnRedirect (821)

24.13.16 TFPHTTPClient.MaxRedirects

Synopsis: Maximum number of redirects to follow automatically

Declaration: `Property MaxRedirects :`

Visibility: published

Access:

Description: Maximum number of HTTP redirects to follow automatically before giving up. Defaults to DefMaxRedirects (805) (16).

See also: AllowRedirect (820), DefMaxRedirects (805)

24.13.17 TFPHTTPClient.OnRedirect

Synopsis: Event called when a redirect is encountered

Declaration: `Property OnRedirect :`

Visibility: published

Access:

Description: Event triggered when an HTTP redirect response is received. The event handler can modify the destination URL or prevent the redirect.

See also: TRedirectEvent (805), AllowRedirect (820)

24.13.18 TFPHTTPClient.UserName

Synopsis: Username for HTTP authentication

Declaration: `Property UserName :`

Visibility: published

Access:

Description: Username to use for HTTP Basic Authentication when the server requires credentials.

See also: Password (821), OnPassword (822)

24.13.19 TFPHTTPClient.Password

Synopsis: Password for HTTP authentication

Declaration: `Property Password :`

Visibility: published

Access:

Description: Password to use for HTTP Basic Authentication when the server requires credentials.

See also: UserName (821), OnPassword (822)

24.13.20 TFPHTTPClient.OnPassword

Synopsis: Event called when authentication is required

Declaration: `Property OnPassword :`

Visibility: published

Access:

Description: Event triggered when the server requests authentication. The event handler can provide credentials and indicate whether to retry the request.

See also: TPasswordEvent (805), UserName (821), Password (821)

24.13.21 TFPHTTPClient.OnDataReceived

Synopsis: Event called during data reception to track progress

Declaration: `Property OnDataReceived :`

Visibility: published

Access:

Description: Event triggered periodically during data reception to provide progress information. Useful for showing download progress to users.

See also: TDataEvent (805), OnDataSent (816)

24.13.22 TFPHTTPClient.OnHeaders

Synopsis: Event called when response headers are received

Declaration: `Property OnHeaders :`

Visibility: published

Access:

Description: Event triggered when HTTP response headers have been received but before the response body is read. Useful for examining headers before processing the response content.

See also: ResponseHeaders (819)

24.13.23 TFPHTTPClient.OnGetSocketHandler

Synopsis: Event called to provide custom socket handlers

Declaration: `Property OnGetSocketHandler :`

Visibility: published

Access:

Description: Event triggered when a socket handler is needed, allowing the application to provide custom socket implementations. This enables the use of custom SSL implementations or socket configurations.

See also: TGetSocketHandlerEvent (805)

24.13.24 TFPHTTPClient.Proxy

Synopsis: Proxy server configuration

Declaration: `Property Proxy :`

Visibility: published

Access:

Description: Configuration for proxy server connections. When configured, all HTTP requests will be routed through the specified proxy server.

See also: TProxyData (824)

24.13.25 TFPHTTPClient.VerifySSLCertificate

Synopsis: Whether to verify SSL certificates

Declaration: `Property VerifySSLCertificate :`

Visibility: published

Access:

Description: When set to `True`, SSL certificates are verified against trusted certificate authorities. When `False`, SSL certificate verification is disabled (less secure).

See also: OnVerifySSLCertificate (823), CertCAFileName (816)

24.13.26 TFPHTTPClient.AfterSocketHandlerCreate

Synopsis: Event called after socket handler creation

Declaration: `Property AfterSocketHandlerCreate :`

Visibility: published

Access:

Description: Event triggered after a socket handler has been created, allowing additional configuration of the handler.

See also: TSocketHandlerCreatedEvent (806), OnGetSocketHandler (822)

24.13.27 TFPHTTPClient.OnVerifySSLCertificate

Synopsis: Event called for custom SSL certificate verification

Declaration: `Property OnVerifySSLCertificate :`

Visibility: published

Access:

Description: Event triggered during SSL certificate verification, allowing custom certificate validation logic. The event handler can accept or reject certificates based on custom criteria.

See also: THTTPVerifyCertificateEvent (805), VerifySSLCertificate (823)

24.14 TProxyData

24.14.1 Description

`TProxyData` stores proxy server configuration including host, port, and authentication credentials. It is used by the HTTP client to route requests through proxy servers.

See also: TFPHTTPClient.Proxy (823)

24.14.2 Method overview

Page	Method	Description
824	Assign	Assigns proxy settings from another object

24.14.3 Property overview

Page	Properties	Access	Description
824	Host	rw	Proxy server hostname or IP address
825	Password	rw	Password for proxy authentication
824	Port	rw	Proxy server port number
825	UserName	rw	Username for proxy authentication

24.14.4 TProxyData.Assign

Synopsis: Assigns proxy settings from another object

Declaration: `procedure Assign(Source: TPersistent);  Override`

Visibility: public

Description: Copies proxy configuration from another `TProxyData` instance.

24.14.5 TProxyData.Host

Synopsis: Proxy server hostname or IP address

Declaration: `Property Host : string`

Visibility: public

Access: Read,Write

Description: The hostname or IP address of the proxy server to use for HTTP requests.

See also: Port (824)

24.14.6 TProxyData.Port

Synopsis: Proxy server port number

Declaration: `Property Port : Word`

Visibility: public

Access: Read,Write

Description: The port number to connect to on the proxy server.

See also: Host (824)

24.14.7 TProxyData.UserName

Synopsis: Username for proxy authentication

Declaration: `Property UserName : string`

Visibility: public

Access: Read,Write

Description: Username to use for proxy server authentication if required.

See also: Password (825)

24.14.8 TProxyData.Password

Synopsis: Password for proxy authentication

Declaration: `Property Password : string`

Visibility: public

Access: Read,Write

Description: Password to use for proxy server authentication if required.

See also: UserName (825)

Chapter 25

Reference for unit 'fpjson'

25.1 Used units

Table 25.1: Used units by unit 'fpjson'

Name	Page
Classes	??
Contnrs	284
System	??
sysutils	??
Variants	??

25.2 Overview

The JSON unit implements JSON support for Free Pascal. It contains the data structures (TJSONData (850) and descendent objects) to treat JSON data and output JSON as a string TJSONData.AsJSON (859). The generated JSON can be formatted in several ways TJSONData.FormatJSON (855).

Using the JSON data structures is simple. Instantiate an appropriate descendent of `TJSONData`, set the data and call `AsJSON`. The following JSON data types are supported:

Numbers in one of TJSONIntegerNumber (862), TJSONFloatNumber (860) or TJSONInt64Number (861), depending on the type of the number.

Strings in TJSONString (877).

Boolean in TJSONBoolean (849).

null is supported using TJSONNull (863)

Array is supported using TJSONArray (838)

Object is supported using TJSONObject (865)

The constructors of these objects allow to set the value, making them very easy to use. The memory management is automatic in the sense that arrays and objects own their values, and when the array or object is freed, all data in it is freed as well.

Typical use would be:

```
Var
  O : TJSONObject;

begin
  O:=TJSONObject.Create(['Age',44,
                         'Firstname','Michael',
                         'Lastname','Van Canneyt']);
  Writeln(O.AsJSON);
  Write('Welcome ',O.Strings['Firstname'],', ');
  Writeln(O.Get('Lastname',''')); // empty default.
  Writeln(', your current age is ',O.Integers('Age']);
  O.Free;
end;
```

The `TJSONArray` and `TJSONObject` classes offer methods to examine, get and set the various members and search through the various members.

Currently the JSON support only allows the use of UTF-8 data.

Parsing incoming JSON and constructing the JSON data structures is not implemented in the `fpJSON` unit. For this, the jsonscanner unit must be included in the program unit clause. This sets several callback hooks (using SetJSONParserHandler (836) and then the GetJSON (834) function can then be used to transform a string or stream to JSON data structures:

```
uses fpjson, jsonparser;

Var
  D,E : TJSONData;

begin
  D:=GetJSON('{ "Children" : ['+
             '  { "Age" : 23, '+
             '    "Names" : { "LastName" : "Rodriquez",'+
             '                "FirstName" : "Roberto" }},'+
             '  { "Age" : 20,'+
             '    "Names" : { "LastName" : "Rodriquez",'+
             '                "FirstName" : "Maria" }}'+
             '  ]}');
  E:=D.FindPath('Children[1].Names.FirstName');
  Writeln(E.AsJSON);
  D.Free;
end.
```

will print "Maria".

The FPJSON code does not use hardcoded class names when creating the JSON: it uses the various CreateJSON (833) functions to create the data. These functions use a registry of classes, so it is possible to create descendents of the classes in the `fpjson` unit and have these used for construction of JSON Data structures. The GetJSONInstanceType (834) and SetJSONInstanceType (836) functions can be used to get or set the classes that must be used. the default parser used by GetJSON (834) will also use these functions.

25.3 Constants, types and variables

25.3.1 Constants

```
ActualValueJSONTypes = ValueJSONTypes - [jtNull]
```

ActualValueJSONTypes is a set constant designating the JSON types that have a non-null single value, i.e., all types except array or object or null.

```
AsCompactJSON = [foSingleLineArray, foSingleLineObject, foskipWhiteSpace
  , foDoNotQuoteMembers]
```

AsCompressedJSON can be used to let FormatJSON (855) behave as TJSONData.AsJSON (859) with TJSONData.CompressedJSON equal to True and TJSONData.UnquotedMemberNames equal to True.

```
AsCompressedJSON = [foSingleLineArray, foSingleLineObject, foskipWhiteSpace
  ]
```

AsCompressedJSON can be used to let TJSONData.FormatJSON (855) behave as TJSONData.AsJSON (859) with TJSONData.CompressedJSON (855) equal to True

```
AsJSONFormat = [foSingleLineArray, foSingleLineObject]
```

AsJSONFormat contains the options that make TJSONData.FormatJSON (855) behave like TJSONData.AsJSON (859)

```
DefaultFormat = []
```

DefaultFormat contains the default formatting options used in formatted JSON.

```
DefaultIndentSize = 2
```

DefaultIndentSize is the default indent size used in formatted JSON.

```
jitNumberLargeInt = jitNumberInt64
```

LargeInt type definition.

```
StructuredJSONTypes = [jtArray, jtObject]
```

StructuredJSONTypes is a set constant designating the JSON types that contain multiple values: array or object.

```
ValueJSONTypes = [jtNumber, jtString, jtBoolean, jtNull]
```

ValueJSONTypes is a set constant designating the JSON types that have a single value, i.e., all types except array or object.

25.3.2 Types

```
PJSONCharType = ^TJSONCharType
```

PJSONCharType is a pointer to a TJSONCharType (830) character. It is used while parsing JSON.

```
TFormatOption = (foSingleLineArray,foSingleLineObject,
  foDoNotQuoteMembers,foUseTabchar,foSkipWhiteSpace,
  foSkipWhiteSpaceOnlyLeading,foForceLF,foFormatFloat)
```

Table 25.2: Enumeration values for type TFormatOption

Value	Explanation
foDoNotQuoteMembers	Do not use quote characters around object member names.
foForceLF	
foFormatFloat	
foSingleLineArray	Keep all array elements on a single line.
foSingleLineObject	Keep all object elements on a single line.
foSkipWhiteSpace	Skip whitespace.
foSkipWhiteSpaceOnlyLeading	Only skip leading whitespace when formatting JSON.
foUseTabchar	Use the tabulator character for indents.

TFormatOption enumerates the various formatting options that can be used in the TJSONData.FormatJSON (855) function.

```
TFormatOptions = Set of TFormatOption
```

TFormatOptions is the set definition used to specify options in TJSONData.FormatJSON (855).

```
TFPJSStream = TMemoryStream
```

TFPJSStream resolves to a stream on native platforms, TJSArray in javascript runtimes.

```
TJSONArrayClass = Class of TJSONArray
```

TJSONArray is the class type for the TJSONArray (838) class. It is used in CreateJSONArray (833).

```
TJSONArrayIterator = procedure(Item: TJSONData; Data: TObject;
  var Continue: Boolean) of object
```

TJSONArrayIterator is the procedural callback used by TJSONArray.Iterate (840) to iterate over the values. Item is the current item in the iteration. Data is the data passed on when calling Iterate. The Continue parameter can be set to false to stop the iteration loop.

```
TJSONBooleanClass = Class of TJSONBoolean
```

TJSONBooleanClass is the class type of TJSONBoolean (849). It is used in the factory methods.

```
TJSONCharType = AnsiChar
```

TJSONCharType is the type of a single character in a TJSONStringType (832) string. It is used by the parser.

```
TJSONDataClass = Class of TJSONData
```

TJSONDataClass is used in the CreateJSON (833), SetJSONInstanceType (836) and GetJSONInstanceType (834) functions to set the actual classes used when creating JSON data.

```
TJSONEnum = record
public
  Key : TJSONStringType;
  KeyNum : Integer
  ;
  Value : TJSONData;
end
```

TJSONEnum is the loop variable type to use when implementing a JSON enumerator (for in). It contains 3 elements which are available in the loop: key, keynum (numerical key) and the actual value (TJSONData).

```
TJSONFloat = Double
```

TJSONFloat is the floating point type used in the JSON support. It is currently a double, but this can be changed easily.

```
TJSONFloatNumberClass = Class of TJSONFloatNumber
```

TJSONFloatNumberClass is the class type of TJSONFloatNumber (860). It is used in the factory methods.

```
TJSONInstanceType = (jitUnknown,jitNumberInteger,jitNumberInt64,
  jitNumberQWord,jitNumberFloat,jitString,jitBoolean
  ,
                     jitNull,jitArray,jitObject)
```

Table 25.3: Enumeration values for type TJSONInstanceType

Value	Explanation
jitArray	Array value.
jitBoolean	Boolean value.
jitNull	Null value.
jitNumberFloat	Floating point real number value.
jitNumberInt64	64-bit signed integer number value.
jitNumberInteger	32-bit signed integer number value.
jitNumberQWord	Qword integer number type.
jitObject	Object value.
jitString	String value.
jitUnknown	Unknown.

TJSONInstanceType is used by the parser to determine what kind of TJSONData (850) descendent to create for a particular data item. It is a more fine-grained division than TJSONType (832)

```
TJSONInt64NumberClass = Class of TJSONInt64Number
```

TJSONInt64NumberClass is the class type of TJSONInt64Number (861). It is used in the factory methods.

```
TJSONIntegerNumberClass = Class of TJSONIntegerNumber
```

TJSONIntegerNumberClass is the class type of TJSONIntegerNumber (862). It is used in the factory methods.

```
TJSONLargeInt = Int64
```

TJSONLargeInt resolves to the largest possible integer type for the current platform. This is NativeInt for Pas2JS and Int64 for all other platforms.

```
TJSONLargeIntNumber = TJSONInt64Number
```

This class is instantiated when a TJSONLargeInt must be represented.

```
TJSONLargeIntNumberClass = TJSONInt64NumberClass
```

TJSONLargeIntNumberClass is the Class reference for TJSONLargeIntNumber (831)

```
TJSONNullClass = Class of TJSONNull
```

TJSONNullClass is the class type of TJSONNull (863). It is used in the factory methods.

```
TJSONNumberType = (ntFloat,ntInteger,ntInt64,ntQWord)
```

Table 25.4: Enumeration values for type TJSONNumberType

Value	Explanation
ntFloat	Floating point value.
ntInt64	64-bit integer value.
ntInteger	32-bit Integer value.
ntQWord	64-bit unsigned integer value.

TJSONNumberType is used to enumerate the different kind of numerical types: JSON only has a single 'number' format. Depending on how the value was parsed, FPC tries to create a value that is as close to the original value as possible: this can be one of integer, int64 or TJSONFloatType (normally a double). The number types have a common ancestor, and they are distinguished by their TJSONNumber.NumberType (865) value.

```
TJSONObjectClass = Class of TJSONObject
```

TJSONObject is the class type for the TJSONObject (865) class. It is used in CreateJSONObject (834).

```
TJSONObjectIterator = procedure(const AName: TJSONStringType;
  Item: TJSONData; Data: TObject;
  var Continue: Boolean) of object
```

TJSONObjectIterator is the procedural callback used by TJSONObject.Iterate (868) to iterate over the values. Item is the current item in the iteration, and AName it's name. Data is the data passed on when calling Iterate. The Continue parameter can be set to false to stop the iteration loop.

```
TJSONParserHandler = procedure(AStream: TStream;
  const AUseUTF8: Boolean;
  out Data: TJSONData)
```

TJSONParserHandler is a callback prototype used by the GetJSON (834) function to do the actual parsing. It has 2 arguments: AStream, which is the stream containing the JSON that must be parsed, and AUseUTF8, which indicates whether the (ansi) strings contain UTF-8.

The result should be returned in Data.

The parser is expected to use the JSON class types registered using the SetJSONInstanceType (836) method, the actual types can be retrieved with GetJSONInstanceType (834)

```
TJSONQWordNumberClass = Class of TJSONQWordNumber
```

TJSONQwordNumberClass is the class type of TJSONQWordNumber (876). It is used in the factory methods.

```
TJSONStringClass = Class of TJSONString
```

TJSONStringClass is the class type of TJSONString (877). It is used in the factory methods.

```
TJSONStringParserHandler = procedure(const aJSON: TJSONStringType
  ;
                                     const AUseUTF8: Boolean;
  out Data: TJSONData)
```

TJSONStringParserHandler is the prototype for the handler to conver a JSON string to a TJSONData (850). It is used in the SetJSONStringParserHandler (836) and GetJSONStringParser-Handler (835) calls.

```
TJSONStringType = UTF8String
```

TJSONFloat is the string point type used in the JSON support. It is currently an ansistring, but this can be changed easily. Unicode characters can be encoded with UTF-8.

```
TJSONtype = (jtUnknown,jtNumber,jtString,jtBoolean,jtNull,jtArray
  ,
             jtObject)
```

Table 25.5: Enumeration values for type TJSONtype

Value	Explanation
jtArray	Array data (integer index,elements can be any type).
jtBoolean	Boolean data.
jtNull	Null data.
jtNumber	Numerical type. This can be integer (32/64 bit) or float.
jtObject	Object data (named index, elements can be any type).
jtString	String data type.
jtUnknown	Unknown JSON data type.

TJSONtype determines the type of JSON data a particular object contains. The class function TJSONData.JSONType (851) returns this type, and indicates what kind of data that particular descendent contains. The values correspond to the original data types in the JSON specification. The TJSONData object itself returns the unknown value.

```
TJSONUnicodeStringType = Unicodestring
```

TJSONUnicodeStringType is an alias used wherever a Unicode (double byte) string is used in the fpJSON code, in particular the TJSONData.AsUnicodeString (857) property.

```
TJSONVariant = variant
```

TJSONVariant resolves to Variant on native platforms, JSValue in javascript runtimes.

25.4 Procedures and functions

25.4.1 CreateJSON

Synopsis: Create a JSON data item.

Declaration:
```
function CreateJSON : TJSONNull
function CreateJSON(Data: Boolean) : TJSONBoolean
function CreateJSON(Data: Integer) : TJSONIntegerNumber
function CreateJSON(Data: Int64) : TJSONInt64Number
function CreateJSON(Data: QWord) : TJSONQWordNumber
function CreateJSON(Data: TJSONFloat) : TJSONFloatNumber
function CreateJSON(const Data: TJSONStringType) : TJSONString
function CreateJSON(const Data: TJSONUnicodeStringType) : TJSONString
```

Visibility: default

Description: CreateJSON will create a JSON Data item depending on the type of data passed to it, and will use the classes returned by GetJSONInstanceType (834) to do so. The classes to be used can be set using the SetJSONInstanceType (836).

The JSON parser uses these functions to create instances of TJSONData (850).

Errors: None.

See also: GetJSONInstanceType (834), SetJSONInstanceType (836), GetJSON (834), CreateJSONArray (833), CreateJSONObject (834)

25.4.2 CreateJSONArray

Synopsis: Create a JSON array.

Declaration:
```
function CreateJSONArray(const Data: Array of const) : TJSONArray
```

Visibility: default

Description: CreateJSONArray retrieves the class registered to represent JSON array data, and creates an instance of this class, passing Data to the constructor. For the Data array the same type conversion rules as for the constructor apply.

Errors: if one of the elements in Data cannot be converted to a JSON structure, an exception will be raised.

See also: GetJSONInstanceType (834), SetJSONInstanceType (836), GetJSON (834), CreateJSON (833), TJSONArray (838)

25.4.3 CreateJSONObject

Synopsis: Create a JSON object.

Declaration:
```
function CreateJSONObject : TJSONObject
function CreateJSONObject(const Data: Array of const) : TJSONObject
```

Visibility: default

Description: `CreateJSONObject` retrieves the class registered to represent JSON object data, and creates an instance of this class, passing `Data` to the constructor. For the `Data` array the same type conversion rules as for the TJSONObject.Create (866) constructor apply.

Errors: if one of the elements in `Data` cannot be converted to a JSON structure, an exception will be raised.

See also: GetJSONInstanceType (834), SetJSONInstanceType (836), GetJSON (834), CreateJSON (833), TJSONObject (865)

25.4.4 GetJSON

Synopsis: Convert JSON string to JSON data structure.

Declaration:
```
function GetJSON(const JSON: TJSONStringType; const UseUTF8: Boolean)
                 : TJSONData
function GetJSON(const JSON: TStream; const UseUTF8: Boolean)
                 : TJSONData
```

Visibility: default

Description: `GetJSON` will read the `JSON` argument (a string or stream that contains a valid JSON data representation) and converts it to native JSON objects. The stream must be positioned on the start of the JSON.

The `fpJSON` unit does not contain a JSON parser. The `jsonparser` unit does contain a JSON parser, and must be included once in the project to be able to parse JSON. The `jsonparser` unit uses the SetJSONParserHandler (836) call to set a callback that is used by `GetJSON` to parse the data.

If `UseUTF8` is set to true, then Unicode characters will be encoded as UTF-8. Otherwise, they are converted to the nearest matching ansi character.

Errors: An exception will be raised if the JSON data stream does not contain valid JSON data.

See also: GetJSONParserHandler (835), SetJSONParserHandler (836), TJSONData (850)

25.4.5 GetJSONInstanceType

Synopsis: JSON factory: Get the JSONData class types to use.

Declaration:
```
function GetJSONInstanceType(AType: TJSONInstanceType) : TJSONDataClass
```

Visibility: default

Description: `GetJSONInstanceType` can be used to retrieve the registered descendents of the TJSONData (850) class, one for each possible kind of data. The result is the class type used to instantiate data of type `AType`.

The JSON parser and the CreateJSON (833) function will use the registered types to instantiate JSON Data. When the parser encounters a value of type `AType`, it will instantiate a class of the type returned by this function. By default, the classes in the `fpJSON` unit are returned.

See also: CreateJSON (833), TJSONData (850), GetJSON (834)

25.4.6 GetJSONParserHandler

Synopsis: Get the current JSON parser handler.

Declaration:
```
function GetJSONParserHandler : TJSONParserHandler
```

Visibility: default

Description: `GetJSONParserHandler` can be used to get the current value of the JSON parser handler callback.

The `fpJSON` unit does not contain a JSON parser in itself: it contains simply the data structure and the ability to write JSON. The parsing must be done using a separate unit.

See also: SetJSONParserHandler (836), GetJSONStringParserHandler (835), TJSONParserHandler (832), GetJSON (834)

25.4.7 GetJSONStringParserHandler

Synopsis: return the current JSON string to JSON Data conversion callback.

Declaration:
```
function GetJSONStringParserHandler : TJSONStringParserHandler
```

Visibility: default

Description: `GetJSONStringParserHandler` returns the handler installed by the last SetJSONStringParserHandler (836) call.

See also: SetJSONStringParserHandler (836), SetJSONParserHandler (836), GetJSONParserHandler (835)

25.4.8 JSONStringToString

Synopsis: Convert a JSON-escaped string to a string.

Declaration:
```
function JSONStringToString(const S: TJSONStringType) : TJSONStringType
function JSONStringToString(const S: UnicodeString) : TJSONStringType
```

Visibility: default

Description: `JSONStringToString` examines the string `S` and replaces any special characters by an escaped string, as in the JSON specification. The following escaped characters are recognized:

```
\\ \" \/ \b \t \n \f \r \u000X
```

See also: StringToJSONString (837), JSONTypeName (835)

25.4.9 JSONTypeName

Synopsis: Convert a JSON type to a string.

Declaration:
```
function JSONTypeName(JSONType: TJSONtype) : string
```

Visibility: default

Description: `JSONTypeName` converts the `JSONType` to a string that describes the type of JSON value.

See also: StringToJSONString (837), JSONStringToString (835)

25.4.10 SetJSONInstanceType

Synopsis: JSON factory: Set the JSONData class types to use.

Declaration:
```
function SetJSONInstanceType(AType: TJSONInstanceType;
                             AClass: TJSONDataClass) : TJSONDataClass
```

Visibility: default

Description: `SetJSONInstanceType` can be used to register descendents of the TJSONData (850) class, one for each possible kind of data. The class type used to instantiate data of type `AType` is passed in `AClass`.

The JSON parser will use the registered types to instantiate JSON Data instanced: when the parser encounters a value of type `AType`, it will instantiate a class of type `AClass`. By default, the classes in the `fpJSON` unit are used.

The CreateJSON (833) functions also use the types registered here to instantiate their data.

The return value is the previously registered instance type for the `AType`.

Errors: If `AClass` is not suitable to contain data of type `AType`, an exception is raised.

See also: GetJSONInstanceType (834), CreateJSON (833)

25.4.11 SetJSONParserHandler

Synopsis: Set the JSON parser handler.

Declaration:
```
function SetJSONParserHandler(AHandler: TJSONParserHandler)
                              : TJSONParserHandler
```

Visibility: default

Description: `SetJSONParserHandler` can be used to set the JSON parser handler callback. The `fpJSON` unit does not contain a JSON parser in itself: it contains simply the data structure and the ability to write JSON. The parsing must be done using a separate unit, and is invoked through a callback. `SetJSONParserHandler` must be used to set this callback.

The `jsonparser` unit does contain a JSON parser, and must be included once in the project to be able to parse JSON. The `jsonparser` unit uses the `SetJSONParserHandler` call to set the callback that is used by `GetJSON` to parse the data. This is done once at the initialization of that unit, so it is sufficient to include the unit in the uses clause of the program.

The function returns the previously registered callback.

This handler uses a stream as input. For speed reasons you can also register handler that converts a string to JSON data. This is done with the SetJSONStringParserHandler (836) call.

See also: SetJSONStringParserHandler (836), GetJSONParserHandler (835), TJSONParserHandler (832), GetJSON (834)

25.4.12 SetJSONStringParserHandler

Synopsis: Install a JSON string to JSON Data conversion callback.

Declaration:
```
function SetJSONStringParserHandler(AHandler: TJSONStringParserHandler)
                                    : TJSONStringParserHandler
```

Visibility: default

Description: `SetJSONStringParserHandler` has the same functionality as SetJSONParserHandler (836). It sets a callback that will be used by the GetJSON (834) call to convert a string value to JSON data. If no such callback is installed, the string will be converted to a stream, and the handler set by SetJSONParserHandler (836) will be used instead. Setting this handler prevents a conversion from a string to a stream.

The function returns the previously installed handler, if any.

See also: GetJSONStringParserHandler (835), SetJSONParserHandler (836), GetJSONParserHandler (835)

25.4.13 StringToJSONString

Synopsis: Convert a string to a JSON-escaped string.

Declaration:
```
function StringToJSONString(const S: TJSONStringType; Strict: Boolean)
                                  : TJSONStringType
function StringToJSONString(const S: UnicodeString; Strict: Boolean)
                                  : TJSONStringType
```

Visibility: default

Description: `StringToJSONString` examines the string `S` and replaces any special characters by an escaped string, as in the JSON specification. The following characters are escaped:

```
\ "  #8 #9 #10 #12 #13.
```

`Strict` indicates that only the absolutely necessary characters will be escaped (when set to `True`) when convering string values to JSON If set to `False/` will also be escaped, although this is strictly speaking not necessary.

See also: JSONStringToString (835), JSONTypeName (835)

25.5 EJSON

25.5.1 Description

`EJSON` is the exception raised by the JSON implementation to report JSON error.

25.6 TBaseJSONEnumerator

25.6.1 Description

`TBaseJSONEnumerator` is the base type for the JSON enumerators. It should not be used directly, instead use the enumerator support of Object pascal to loop over values in JSON data.

The value of the `TBaseJSONEnumerator` enumerator is a record that describes the key and value of a JSON value. The key can be string-based (for records) or numerical (for arrays).

See also: TJSONEnum (830)

25.6.2 Method overview

Page	Method	Description
838	GetCurrent	Return the current value of the enumerator.
838	MoveNext	Move to next value in array/object.

25.6.3 Property overview

Page	Properties	Access	Description
838	Current	r	Return the current value of the enumerator.

25.6.4 TBaseJSONEnumerator.GetCurrent

Synopsis: Return the current value of the enumerator.

Declaration: `function GetCurrent : TJSONEnum; Virtual; Abstract`

Visibility: public

Description: `GetCurrent` returns the current value of the enumerator. This is a TJSONEnum (830) value.

See also: TJSONEnum (830)

25.6.5 TBaseJSONEnumerator.MoveNext

Synopsis: Move to next value in array/object.

Declaration: `function MoveNext : Boolean; Virtual; Abstract`

Visibility: public

Description: `MoveNext` attempts to move to the next value. This will return `True` if the move was successful, or `False` if not. When `True` is returned, then

See also: TJSONEnum (830), TJSONData (850)

25.6.6 TBaseJSONEnumerator.Current

Synopsis: Return the current value of the enumerator.

Declaration: `Property Current : TJSONEnum`

Visibility: public

Access: Read

Description: `Current` returns the current enumerator value of type TJSONEnum (830). It is only valid after MoveNext (838) returned `True`.

See also: TJSONEnum (830), TJSONData (850), MoveNext (838)

25.7 TJSONArray

25.7.1 Description

`TJSONArrayClass` is the class type of TJSONArray (838). It is used in the factory methods.

See also: TJSONArray (838), SetJSONInstanceType (836), GetJSONInstanceType (834)

25.7.2 Method overview

Page	Method	Description
841	Add	Add a JSON value to the array.
841	Clear	Clear the array.
840	Clone	Clone the JSON array.
839	Create	Create a new instance of JSON array data.
842	Delete	Delete an element from the list by index.
840	Destroy	Free the JSON array.
842	Exchange	Exchange 2 elements in the list.
842	Extract	Extract an element from the array.
841	GetEnumerator	Get an array enumerator.
841	IndexOf	Return index of JSONData instance in array.
843	Insert	Insert an element in the array.
840	Iterate	Iterate over all elements in the array.
840	JSONType	native JSON data type.
843	Move	Move a value from one location to another.
843	Remove	Remove an element from the list.
844	Sort	Sort the items in the array.

25.7.3 Property overview

Page	Properties	Access	Description
848	Arrays	rw	Get or set elements as JSON array values.
848	Booleans	rw	Get or set elements as boolean values.
847	Floats	rw	Get or set elements as floating-point numerical values.
845	Int64s	rw	Get or set elements as Int64 values.
845	Integers	rw	Get or set elements as integer values.
844	Items		Indexed access to the values in the array.
846	LargeInts	rw	Get or set elements as LargeInt values.
844	Nulls	r	Check which elements are null.
848	Objects	rw	Get or set elements as JSON object values.
846	QWords	rw	Get or set elements as QWord values.
847	Strings	rw	Get or set elements as string values.
844	Types	r	JSON types of elements in the array.
846	UnicodeStrings	rw	Get or set elements as Unicode string values.

25.7.4 TJSONArray.Create

Synopsis: Create a new instance of JSON array data.

Declaration:
```
constructor Create;  Overload;  Reintroduce
constructor Create(const Elements: Array of const);  Overload
```

Visibility: public

Description: `Create` creates a new JSON array instance, and initializes the data with `Elements`. The elements are converted to various TJSONData (850) instances, instances of `TJSONData` are inserted in the array as-is.

The data type of the inserted objects is determined from the type of data passed to it, with a natural mapping. A `Nil` pointer will be inserted as a `TJSONNull` value.

Errors: If an invalid class or not recognized data type (pointer) is inserted in the elements array, an `EConvertError` exception will be raised.

See also: GetJSONInstanceType (834)

25.7.5 TJSONArray.Destroy

Synopsis: Free the JSON array.

Declaration: `destructor Destroy;  Override`

Visibility: public

Description: `Destroy` will delete all elements in the array and clean up the TJSONArray (838) instance.

See also: TJSONArray.Clear (841), TJSONArray.Create (839)

25.7.6 TJSONArray.JSONType

Synopsis: native JSON data type.

Declaration: `class function JSONType : TJSONtype;  Override`

Visibility: public

Description: `JSONType` is overridden by `TJSONArray` to return `jtArray`.

See also: TJSONData.JSONType (851)

25.7.7 TJSONArray.Clone

Synopsis: Clone the JSON array.

Declaration: `function Clone : TJSONData;  Override`

Visibility: public

Description: `Clone` creates a new `TJSONArray`, clones all elements in the array and adds them to the newly created array in the same order as they are in the array.

See also: TJSONData.Clone (855)

25.7.8 TJSONArray.Iterate

Synopsis: Iterate over all elements in the array.

Declaration: `procedure Iterate(Iterator: TJSONArrayIterator; Data: TObject)`

Visibility: public

Description: `Iterate` iterates over all elements in the array, passing them one by one to the `Iterator` callback, together with the `Data` parameter. The iteration stops when all elements have been passed or when the iterator callback returned `False` in the `Continue` parameter.

See also: TJSONArrayIterator (829)

25.7.9 TJSONArray.IndexOf

Synopsis: Return index of JSONData instance in array.

Declaration:
```
function IndexOf(obj: TJSONData) : Integer
```

Visibility: public

Description: `IndexOf` compares all elements in the array with `Obj` and returns the index of the element instance that equals `Obj`. The actual instances are compared, not the JSON value. If none of the elements match, the function returns -1.

See also: Clear (841)

25.7.10 TJSONArray.GetEnumerator

Synopsis: Get an array enumerator.

Declaration:
```
function GetEnumerator : TBaseJSONEnumerator;  Override
```

Visibility: public

Description: `GetEnumerator` is overridden in `TJSONarray` so it returns an array enumerator. The array enumerator will return all the elements in the array, and stores their index in the `KeyNum` member of TJSONEnum (830).

See also: TJSONEnum (830), TJSONData.GetEnumerator (852)

25.7.11 TJSONArray.Clear

Synopsis: Clear the array.

Declaration:
```
procedure Clear;  Override
```

Visibility: public

Description: `Clear` clears the array and frees all elements in it. After the call to clear, Count (856) returns 0.

See also: Delete (842), Extract (842)

25.7.12 TJSONArray.Add

Synopsis: Add a JSON value to the array.

Declaration:
```
function Add(Item: TJSONData) : Integer
function Add(I: Integer) : Integer
function Add(I: Int64) : Int64
function Add(I: QWord) : QWord
function Add(const S: UnicodeString) : Integer
function Add(const S: TJSONStringType) : Integer
function Add : Integer
function Add(F: TJSONFloat) : Integer
function Add(B: Boolean) : Integer
function Add(AnArray: TJSONArray) : Integer
function Add(AnObject: TJSONObject) : Integer
```

Visibility: public

Description: `Add` adds the value passed on to the array. If it is a plain pascal value, it is converted to an appropriate TJSONData (850) instance. If a `TJSONData` instance is passed, it is simply added to the array. Note that the instance will be owned by the array, and destroyed when the array is cleared (this is in particular true is an JSON array or object).

The function returns the `TJSONData` instance that was added to the array.

See also: Delete (842), Extract (842)

25.7.13 TJSONArray.Delete

Synopsis: Delete an element from the list by index.

Declaration:
```
procedure Delete(Index: Integer)
```

Visibility: public

Description: `Delete` deletes the element with given `Index` from the list. The TJSONData (850) element is freed.

Errors: If an invalid index is passed, an exception is raised.

See also: Clear (841), Add (841), Extract (842), Exchange (842)

25.7.14 TJSONArray.Exchange

Synopsis: Exchange 2 elements in the list.

Declaration:
```
procedure Exchange(Index1: Integer; Index2: Integer)
```

Visibility: public

Description: `Exchange` exchanges 2 elements at locations `Index1` and `Index2` in the list. This is more efficient than manually extracting and adding the elements to the list.

Errors: If an invalid index (for either element) is passed, an exception is raised.

25.7.15 TJSONArray.Extract

Synopsis: Extract an element from the array.

Declaration:
```
function Extract(Item: TJSONData) : TJSONData
function Extract(Index: Integer) : TJSONData
```

Visibility: public

Description: `Extract` removes the element at position `Index` or the indicated element from the list, just as Delete (842) does. In difference with `Delete`, it does not free the object instance. Instead, it returns the extracted element.

See also: Delete (842), Clear (841), Insert (843), Add (841)

25.7.16 TJSONArray.Insert

Synopsis: Insert an element in the array.

Declaration:
```
procedure Insert(Index: Integer)
procedure Insert(Index: Integer; Item: TJSONData)
procedure Insert(Index: Integer; I: Integer)
procedure Insert(Index: Integer; I: Int64)
procedure Insert(Index: Integer; I: QWord)
procedure Insert(Index: Integer; const S: UnicodeString)
procedure Insert(Index: Integer; const S: TJSONStringType)
procedure Insert(Index: Integer; F: TJSONFloat)
procedure Insert(Index: Integer; B: Boolean)
procedure Insert(Index: Integer; AnArray: TJSONArray)
procedure Insert(Index: Integer; AnObject: TJSONObject)
```

Visibility: public

Description: `Insert` adds a value or element to the array at position `Index`. Elements with index equal to or larger than `Index` are shifted. Like Add (841), it converts plain pascal values to JSON values.

Note that when inserting a TJSONData (850) instance to the array, it is owned by the array. `Index` must be a value between 0 and `Count-1`.

Errors: If an invalid index is specified, an exception is raised.

See also: Add (841), Delete (842), Extract (842), Clear (841)

25.7.17 TJSONArray.Move

Synopsis: Move a value from one location to another.

Declaration:
```
procedure Move(CurIndex: Integer; NewIndex: Integer)
```

Visibility: public

Description: `Move` moves the element at index `CurIndex` to the position `NewIndex`. It will shift the elements in between as needed. This operation is more efficient than extracting and inserting the element manually.

See also: Exchange (842), Extract (842), Insert (843)

25.7.18 TJSONArray.Remove

Synopsis: Remove an element from the list.

Declaration:
```
procedure Remove(Item: TJSONData)
```

Visibility: public

Description: `Remove` removes `item` from the array, if it is in the array. The object pointer is checked for presence in the array, not the JSON values. Note that the element is freed if it was in the array and is removed.

See also: Delete (842), Extract (842)

25.7.19 TJSONArray.Sort

Synopsis: Sort the items in the array.

Declaration: `procedure Sort(Compare: TListSortCompare)`

Visibility: public

Description: `Sort` can be used to perform a sort in an array. The array does not compare elements, for this the `Compare` callback must be used, to compare 2 elements from the array.

Errors: None.

See also: #rtl.classes.TListSortCompare (**??**)

25.7.20 TJSONArray.Items

Synopsis: Indexed access to the values in the array.

Declaration: `Property Items : ; default`

Visibility: public

Access:

Description: `Items` is introduced in TJSONData.Items (856). `TJSONArray` simply declares it as the default property.

See also: TJSONData.Items (856)

25.7.21 TJSONArray.Types

Synopsis: JSON types of elements in the array.

Declaration: `Property Types[Index: Integer]: TJSONtype`

Visibility: public

Access: Read

Description: `Types` gives direct access to the TJSONData.JSONType (851) result of the elements in the array. Accessing it is equivalent to accessing

```
Items[Index].JSONType
```

See also: TJSONData.JSONType (851), TJSONData.Items (856)

25.7.22 TJSONArray.Nulls

Synopsis: Check which elements are null.

Declaration: `Property Nulls[Index: Integer]: Boolean`

Visibility: public

Access: Read

Description: `Nulls` gives direct access to the TJSONData.IsNull (859) property when reading. It is then equivalent to accessing

```
Items[Index].IsNull
```

See also: TJSONData.JSONType (851), TJSONData.Items (856), TJSONData.IsNull (859), TJSONArray.Types (844)

25.7.23 TJSONArray.Integers

Synopsis: Get or set elements as integer values.

Declaration: `Property Integers[Index: Integer]: Integer`

Visibility: public

Access: Read,Write

Description: `Integers` gives direct access to the TJSONData.AsInteger (858) property when reading. Reading it is the equivalent to accessing

```
Items[Index].AsInteger
```

When writing, it will check if an integer JSON value is located at the given location, and replace it with the new value. If a non-integer JSON value is there, it is replaced with the written integer value.

See also: TJSONData.Items (856), TJSONData.IsNull (859), TJSONArray.Types (844), TJSONArray.Int64s (845), TJSONArray.QWords (846), TJSONArray.Floats (847), TJSONArray.Strings (847), TJSONArray.Booleans (848)

25.7.24 TJSONArray.Int64s

Synopsis: Get or set elements as Int64 values.

Declaration: `Property Int64s[Index: Integer]: Int64`

Visibility: public

Access: Read,Write

Description: `Int64s` gives direct access to the TJSONData.AsInt64 (857) property when reading. Reading it is the equivalent to accessing

```
Items[Index].AsInt64
```

When writing, it will check if an 64-bit integer JSON value is located at the given location, and replace it with the new value. If a non-64-bit-integer JSON value is there, it is replaced with the written int64 value.

See also: TJSONData.Items (856), TJSONData.IsNull (859), TJSONArray.Types (844), TJSONArray.Integers (845), TJSONArray.Floats (847), TJSONArray.Strings (847), TJSONArray.Booleans (848), TJSONArray.QWords (846)

25.7.25 TJSONArray.LargeInts

Synopsis: Get or set elements as LargeInt values.

Declaration: `Property LargeInts[Index: Integer]: TJSONLargeInt`

Visibility: public

Access: Read,Write

Description: `LargeInts` gives direct access to the TJSONData.AsLargeInt (858) property when reading. Reading it is the equivalent to accessing

```
Items[Index].AsLargeInt
```

When writing, it will check if an Largeint integer JSON value is located at the given location, and replace it with the new value. If a non-large-integer JSON value is there, it is replaced with the written value.

See also: TJSONData.AsLargeInt (858), TJSONData.Items (856), TJSONData.IsNull (859), TJSONArray.Types (844), TJSONArray.Integers (845), TJSONArray.Floats (847), TJSONArray.Strings (847), TJSONArray.Booleans (848), TJSONArray.QWords (846)

25.7.26 TJSONArray.QWords

Synopsis: Get or set elements as QWord values.

Declaration: `Property QWords[Index: Integer]: QWord`

Visibility: public

Access: Read,Write

Description: `QWords` gives direct access to the AsQWord (857) property when reading. Reading it is the equivalent to accessing

```
Items[Index].AsQWord
```

When writing, it will check if an 64-bit unsigned integer JSON value is located at the given location, and replace it with the new value. If a non-64-bit unsigned integer JSON value is there, it is replaced with the written QWord value.

See also: Items (826), AsQWord (857), IsNull (859), Types (844), Integers (845), Floats (847), Strings (847), Booleans (848)

25.7.27 TJSONArray.UnicodeStrings

Synopsis: Get or set elements as Unicode string values.

Declaration: `Property UnicodeStrings[Index: Integer]: TJSONUnicodeStringType`

Visibility: public

Access: Read,Write

Description: `UNicodeStrings` gives direct access to the TJSONData.AsUnicodeString (857) property when reading. Reading it is the equivalent to accessing

```
Items[Index].AsUnicodeString
```

When writing, it will check if a UNicodeStrings JSON value is located at the given location, and replace it with the new value. If a non-string value is there, it is replaced with the written Unicode string value.

See also: TJSONData.Items (856), TJSONData.IsNull (859), TJSONArray.Types (844), TJSONArray.Integers (845), TJSONArray.QWords (846), TJSONArray.Floats (847), TJSONArray.Int64s (845), TJSONArray.Booleans (848), TJSONArray.Strings (847)

25.7.28 TJSONArray.Strings

Synopsis: Get or set elements as string values.

Declaration: `Property Strings[Index: Integer]: TJSONStringType`

Visibility: public

Access: Read,Write

Description: `Strings` gives direct access to the TJSONData.AsString (856) property when reading. Reading it is the equivalent to accessing

```
Items[Index].AsString
```

When writing, it will check if a string JSON value is located at the given location, and replace it with the new value. If a non-string value is there, it is replaced with the written string value.

See also: TJSONData.Items (856), TJSONData.IsNull (859), TJSONArray.Types (844), TJSONArray.Integers (845), TJSONArray.QWords (846), TJSONArray.Floats (847), TJSONArray.Int64s (845), TJSONArray.Booleans (848)

25.7.29 TJSONArray.Floats

Synopsis: Get or set elements as floating-point numerical values.

Declaration: `Property Floats[Index: Integer]: TJSONFloat`

Visibility: public

Access: Read,Write

Description: `Floats` gives direct access to the TJSONData.AsFloat (858) property when reading. Reading it is the equivalent to accessing

```
Items[Index].AsFloat
```

When writing, it will check if a floating point numerical JSON value is located at the given location, and replace it with the new value. If a non-floating point numerical value is there, it is replaced with the written floating point value.

See also: TJSONData.Items (856), TJSONData.IsNull (859), TJSONArray.Types (844), TJSONArray.Integers (845), TJSONArray.Strings (847), TJSONArray.Int64s (845), TJSONArray.QWords (846), TJSONArray.Booleans (848)

25.7.30 TJSONArray.Booleans

Synopsis: Get or set elements as boolean values.

Declaration: `Property Booleans[Index: Integer]: Boolean`

Visibility: public

Access: Read,Write

Description: `Floats` gives direct access to the TJSONData.AsBoolean (859) property when reading. Reading it is the equivalent to accessing

```
Items[Index].AsBoolean
```

When writing, it will check if a boolean JSON value is located at the given location, and replace it with the new value. If a non-boolean value is there, it is replaced with the written boolean value.

See also: TJSONData.Items (856), TJSONData.IsNull (859), TJSONArray.Types (844), TJSONArray.Integers (845), TJSONArray.Strings (847), TJSONArray.Int64s (845), TJSONArray.QWords (846), TJSONArray.Floats (847)

25.7.31 TJSONArray.Arrays

Synopsis: Get or set elements as JSON array values.

Declaration: `Property Arrays[Index: Integer]: TJSONArray`

Visibility: public

Access: Read,Write

Description: `Arrays` gives direct access to JSON Array values when reading. Reading it is the equivalent to accessing

```
Items[Index] As TJSONArray
```

When writing, it will replace any previous value at that location with the written value. Note that the old value is freed, and the new value is owned by the array.

See also: TJSONData.Items (856), TJSONData.IsNull (859), TJSONArray.Types (844), TJSONArray.Integers (845), TJSONArray.Strings (847), TJSONArray.Int64s (845), TJSONArray.QWords (846), TJSONArray.Floats (847), TJSONArray.Objects (848)

25.7.32 TJSONArray.Objects

Synopsis: Get or set elements as JSON object values.

Declaration: `Property Objects[Index: Integer]: TJSONObject`

Visibility: public

Access: Read,Write

Description: `Objects` gives direct access to JSON object values when reading. Reading it is the equivalent to accessing

```
Items[Index] As TJSONObject
```

When writing, it will replace any previous value at that location with the whiten value. Note that the old value is freed, and the new value is owned by the array.

See also: TJSONData.Items (856), TJSONData.IsNull (859), TJSONArray.Types (844), TJSONArray.Integers (845), TJSONArray.Strings (847), TJSONArray.Int64s (845), TJSONArray.QWords (846), TJSONArray.Floats (847), TJSONArray.Arrays (848)

25.8 TJSONBoolean

25.8.1 Description

`TJSONBoolean` must be used whenever boolean data must be represented. It has limited functionality to convert the value from or to integer or floating point data.

See also: TJSONFloatNumber (860), TJSONIntegerNumber (862), TJSONInt64Number (861), TJSONBoolean (849), TJSONNull (863), TJSONArray (838), TJSONObject (865)

25.8.2 Method overview

Page	Method	Description
850	Clear	Clear data.
850	Clone	Clone boolean value.
849	Create	Create a new instance of boolean JSON data.
849	JSONType	native JSON data type.

25.8.3 TJSONBoolean.Create

Synopsis: Create a new instance of boolean JSON data.

Declaration: `constructor Create(AValue: Boolean);  Reintroduce`

Visibility: public

Description: `Create` instantiates a new boolean JSON data and initializes the value with `AValue`.

See also: TJSONIntegerNumber.Create (863), TJSONFloatNumber.Create (860), TJSONInt64Number.Create (861), TJSONString.Create (877), TJSONArray.Create (839), TJSONObject.Create (866)

25.8.4 TJSONBoolean.JSONType

Synopsis: native JSON data type.

Declaration: `class function JSONType : TJSONtype;  Override`

Visibility: public

Description: `JSONType` is overridden by `TJSONString` to return `jtBoolean`.

See also: TJSONData.JSONType (851)

25.8.5 TJSONBoolean.Clear

Synopsis: Clear data.

Declaration: `procedure Clear;  Override`

Visibility: public

Description: `Clear` is overridden by `TJSONBoolean` to set the value to `False`.

See also: TJSONData.Clear (852)

25.8.6 TJSONBoolean.Clone

Synopsis: Clone boolean value.

Declaration: `function Clone : TJSONData;  Override`

Visibility: public

Description: `Clone` overrides TJSONData.Clone (855) and creates an instance of the same class with the same boolean value.

See also: TJSONData.Clone (855)

25.9 TJSONData

25.9.1 Description

`TJSONData` is an abstract class which introduces all properties and methods needed to work with JSON-based data. It should never be instantiated. Based on the type of data that must be represented one of the following descendents must be instantiated instead.

Numbers must be represented using one of TJSONIntegerNumber (862), TJSONFloatNumber (860) or TJSONInt64Number (861), depending on the type of the number.

Strings can be represented with TJSONString (877).

Boolean can be represented with TJSONBoolean (849).

null is supported using TJSONNull (863)

Array data can be represented using TJSONArray (838)

Object data can be supported using TJSONObject (865)

See also: TJSONIntegerNumber (862), TJSONString (877), TJSONBoolean (849), TJSONNull (863), TJSONArray (838), TJSONObject (865)

25.9.2 Method overview

Page	Method	Description
852	Clear	Clear the raw value of this data object.
855	Clone	Duplicate the value of the JSON data.
851	Create	Create a new instance of TJSONData.
852	DumpJSON	Fast, memory efficient dump of JSON in stream.
852	FindPath	Find data by name.
855	FormatJSON	Return a formatted JSON representation of the data.
852	GetEnumerator	Return an enumerator for the data.
854	GetPath	Get data by name.
851	JSONType	The native JSON data type represented by this object.

25.9.3 Property overview

Page	Properties	Access	Description
859	AsBoolean	rw	Access the raw JSON value as a boolean.
858	AsFloat	rw	Access the raw JSON value as a float.
857	AsInt64	rw	Access the raw JSON value as an 64-bit integer.
858	AsInteger	rw	Access the raw JSON value as an 32-bit integer.
859	AsJSON	r	Return a JSON representation of the value.
858	AsLargeInt	rw	Access to data as largeint.
857	AsQWord	rw	Access the raw JSON value as an 64-bit unsigned integer.
856	AsString	rw	Access the raw JSON value as a string.
857	AsUnicodeString	rw	Return the value as a Unicode string.
855	CompressedJSON	rw	Compress JSON - skip whitespace.
856	Count	r	Number of sub-items for this data element.
859	IsNull	r	Is the data a null value ?
856	Items	rw	Indexed access to sub-items.
856	Value	rw	The value of this data object as a variant.

25.9.4 TJSONData.JSONType

Synopsis: The native JSON data type represented by this object.

Declaration: `class function JSONType : TJSONtype;  Virtual`

Visibility: public

Description: `JSONType` indicates the JSON data type that this object will be written as, or the JSON data type that instantiated this object. In `TJSONData`, this function returns `jtUnknown`. Descendents override this method to return the correct data type.

See also: TJSONtype (832)

25.9.5 TJSONData.Create

Synopsis: Create a new instance of TJSONData.

Declaration: `constructor Create;  Virtual`

Visibility: public

Description: `Create` instantiates a new `TJSONData` object. It should never be called directly, instead one of the descendents should be instantiated.

See also: TJSONIntegerNumber.Create (863), TJSONString.Create (877), TJSONBoolean.Create (849), TJSONArray.Create (839), TJSONObject.Create (866)

25.9.6 TJSONData.Clear

Synopsis: Clear the raw value of this data object.

Declaration: `procedure Clear;  Virtual;  Abstract`

Visibility: public

Description: `Clear` is implemented by the descendents of `TJSONData` to clear the data. An array will be emptied, an object will remove all properties, numbers are set to zero, strings set to the empty string, etc.

See also: Create (851)

25.9.7 TJSONData.DumpJSON

Synopsis: Fast, memory efficient dump of JSON in stream.

Declaration: `procedure DumpJSON(S: TFPJSStream)`

Visibility: public

Description: `DumpJSON` writes the data as a JSON string to the stream `S`. No intermediate strings are created, making this a more fast and memory efficient operation than creating a string with TJSONData.AsJSON (859) and writing it to stream.

Errors: None.

See also: TJSONData.FormatJSON (855), TJSONData.AsJSON (859)

25.9.8 TJSONData.GetEnumerator

Synopsis: Return an enumerator for the data.

Declaration: `function GetEnumerator : TBaseJSONEnumerator;  Virtual`

Visibility: public

Description: `GetEnumerator` returns an enumerator for the JSON data. For simple types, the enumerator will just contain the current value. For arrays and objects, the enumerator will loop over the values in the array. The return value is not a TJSONData (850) type, but a TJSONEnum (830) structure, which contains the value, and for structured types, the key (numerical or string).

See also: TJSONEnum (830), TJSONArray (838), TJSONObject (865)

25.9.9 TJSONData.FindPath

Synopsis: Find data by name.

Declaration: `function FindPath(const APath: TJSONStringType) : TJSONData`

Visibility: public

Description: `FindPath` finds a value based on its path. If none is found, `Nil` is returned. The path elements are separated by dots and square brackets, as in object member notation or array notation. The path is case sensitive.

- For simple values, the path must be empty.
- For JSON objects (865), a member can be specified using its name, and the object value itself can be retrieved with the empty path.
- For JSON Arrays (865), the elements can be found based on an array index. The array value itself can be retrieved with the empty path.

The following code will return the value itself, i.e. `E` will contain the same element as D:

```
Var
  D,E : TJSONData;

begin
  D:=TJSONIntegerNumber.Create(123);
  E:=D.FindPath('');
end.
```

The following code will not return anything:

```
Var
  D,E : TJSONData;

begin
  D:=TJSONIntegerNumber.Create(123);
  E:=D.FindPath('a');
end.
```

The following code will return the third element from the array:

```
Var
  D,E : TJSONData;

begin
  D:=TJSONArray.Create([1,2,3,4,5]);
  E:=D.FindPath('[2]');
  Writeln(E.AsJSON);
end.
```

The output of this program is 3.

The following code returns the element `Age` from the object:

```
Var
  D,E : TJSONData;

begin
  D:=TJSONObject.Create(['Age',23,
                         'Lastame','Rodriguez',
                         'FirstName','Roberto']);
  E:=D.FindPath('Age');
  Writeln(E.AsJSON);
end.
```

The code will print 23.

Obviously, this can be combined:

```
Var
  D,E : TJSONData;

begin
  D:=TJSONObject.Create(['Age',23,
                         'Names', TJSONObject.Create([
                             'LastName','Rodriguez',
                             'FirstName','Roberto'])]);
  E:=D.FindPath('Names.LastName');
  Writeln(E.AsJSON);
end.
```

And mixed:

```
var
  D,E : TJSONData;

begin
  D:=TJSONObject.Create(['Children',
       TJSONArray.Create([
         TJSONObject.Create(['Age',23,
                             'Names', TJSONObject.Create([
                               'LastName','Rodriguez',
                               'FirstName','Roberto'])
         ]),
         TJSONObject.Create(['Age',20,
                             'Names', TJSONObject.Create([
                                   'LastName','Rodriguez',
                                   'FirstName','Maria'])
        ])
      ])
     ]);
  E:=D.FindPath('Children[1].Names.FirstName');
  Writeln(E.AsJSON);
end.
```

See also: TJSONArray (838), TJSONObject (865), GetPath (854)

25.9.10 TJSONData.GetPath

Synopsis: Get data by name.

Declaration: `function GetPath(const APath: TJSONStringType) : TJSONData`

Visibility: public

Description: `GetPath` is identical to FindPath (852) but raises an exception if no element was found. The exception message contains the piece of path that was not found.

Errors: An EJSON (837) exception is raised if the path does not exist.

See also: FindPath (852)

25.9.11 TJSONData.Clone

Synopsis: Duplicate the value of the JSON data.

Declaration:
```
function Clone : TJSONData;  Virtual;  Abstract
```

Visibility: public

Description: `Clone` returns a new instance of the `TJSONData` descendent that has the same value as the instance, i.e. the `AsJSON` property of the instance and its clone is the same.

Note that the clone must be freed by the caller. Freeing a JSON object will not free its clones.

Errors: Normally, no JSON-specific errors should occur, but an EOutOfMemory (**??**) exception can be raised.

See also: Clear (852), EOutOfMemory (**??**)

25.9.12 TJSONData.FormatJSON

Synopsis: Return a formatted JSON representation of the data.

Declaration:
```
function FormatJSON(Options: TFormatOptions; Indentsize: Integer)
                          : TJSONStringType
```

Visibility: public

Description: `FormatJSON` returns a formatted JSON representation of the data. For simple JSON values, this is the same representation as the AsJSON (826) property, but for complex values (TJSONArray (838) and TJSONObject (865)) the JSON is formatted differently.

There are some optional parameters to control the formatting. `Options` controls the use of whitespace and newlines. `IndentSize` controls the amount of indent applied when starting a new line.

The implementation is not optimized for speed.

See also: AsJSON (859), TFormatOptions (829)

25.9.13 TJSONData.CompressedJSON

Synopsis: Compress JSON - skip whitespace.

Declaration:
```
Property CompressedJSON : Boolean
```

Visibility: public

Access: Read,Write

Description: `CompressedJSON` can be used to let TJSONData.AsJSON (859) return JSON which does not contain any whitespace. By default it is `False` and whitespace is inserted. If set to `True`, output will contain no whitespace.

See also: TJSONData.FormatJSON (855), TJSONData.AsJSON (859), TJSONData.UnquotedMemberNames (850), TJSONData.AsCompressedJSON (850), TJSONObject.UnquotedMemberNames (871)

25.9.14 TJSONData.Count

Synopsis: Number of sub-items for this data element.

Declaration: `Property Count : Integer`

Visibility: public

Access: Read

Description: `Count` is the amount of members of this data element. For simple values (null, boolean, number and string) this is zero. For complex structures, this is the amount of elements in the array or the number of properties of the object

See also: Items (856)

25.9.15 TJSONData.Items

Synopsis: Indexed access to sub-items.

Declaration: `Property Items[Index: Integer]: TJSONData`

Visibility: public

Access: Read,Write

Description: `Items` allows indexed access to the sub-items of this data. The `Index` is 0-based, and runs from 0 to `Count-1`. For simple data types, this function always returns `Nil`, the complex data type descendents (TJSONArray (838) and TJSONObject (865)) override this method to return the `Index`-th element in the list.

See also: Count (856), TJSONArray (838), TJSONObject (865)

25.9.16 TJSONData.Value

Synopsis: The value of this data object as a variant.

Declaration: `Property Value : TJSONVariant`

Visibility: public

Access: Read,Write

Description: `Value` returns the value of the data object as a variant when read, and converts the variant value to the native JSON type of the object. It does not change the native JSON type (TJSONType (832)), so the variant value must be convertible to the native JSON type.

For complex types, reading or writing this property will raise an `EConvertError` exception.

See also: TJSONType (832)

25.9.17 TJSONData.AsString

Synopsis: Access the raw JSON value as a string.

Declaration: `Property AsString : TJSONStringType`

Visibility: public

Access: Read,Write

Description: `AsString` allows access to the raw value as a string. When reading, it converts the native value of the data to a string. When writing, it attempts to transform the string to a native value. If this conversion fails, an `EConvertError` exception is raised.

For TJSONString (877) this will return the native value.

For complex values, reading or writing this property will result in an `EConvertError` exception.

See also: AsInteger (858), Value (856), AsInt64 (857), AsFloat (858), AsBoolean (859), IsNull (859), AsJSON (859)

25.9.18 TJSONData.AsUnicodeString

Synopsis: Return the value as a Unicode string.

Declaration: `Property AsUnicodeString : TJSONUnicodeStringType`

Visibility: public

Access: Read,Write

Description: `AsUnicodeString` returns the value of a simple JSON value as a Unicode string.

See also: TJSONData.AsString (856)

25.9.19 TJSONData.AsInt64

Synopsis: Access the raw JSON value as an 64-bit integer.

Declaration: `Property AsInt64 : Int64`

Visibility: public

Access: Read,Write

Description: `AsInt64` allows access to the raw value as a 64-bit integer value. When reading, it attempts to convert the native value of the data to a 64-bit integer value. When writing, it attempts to transform the 64-bit integer value to a native value. If either conversion fails, an `EConvertError` exception is raised.

For TJSONInt64Number (861) this will return the native value.

For complex values, reading or writing this property will always result in an `EConvertError` exception.

See also: AsFloat (858), Value (856), AsInteger (858), AsString (856), AsBoolean (859), IsNull (859), AsJSON (859)

25.9.20 TJSONData.AsQWord

Synopsis: Access the raw JSON value as an 64-bit unsigned integer.

Declaration: `Property AsQWord : QWord`

Visibility: public

Access: Read,Write

Description: `AsQWord` allows access to the raw value as a 64-bit unsigned integer value. When reading, it attempts to convert the native value of the data to a 64-bit unsigned integer value. When writing, it attempts to transform the 64-bit unsigned integer value to a native value. If either conversion fails, an `EConvertError` exception is raised.

For TJSONQwordNumber (876) this will return the native value.

For complex values, reading or writing this property will always result in an `EConvertError` exception.

See also: AsFloat (858), Value (856), AsInteger (858), AsInt64 (857), AsString (856), AsBoolean (859), IsNull (859), AsJSON (859)

25.9.21 TJSONData.AsLargeInt

Synopsis: Access to data as largeint.

Declaration: `Property AsLargeInt : TJSONLargeInt`

Visibility: public

Access: Read,Write

Description: `AsLargeInt` returns an integer value of the largest possible integer type for the current platform: This is `NativeInt` on the pas2JS platform, `Int64` on all other platforms.

25.9.22 TJSONData.AsFloat

Synopsis: Access the raw JSON value as a float.

Declaration: `Property AsFloat : TJSONFloat`

Visibility: public

Access: Read,Write

Description: `AsFloat` allows access to the raw value as a floating-point value. When reading, it converts the native value of the data to a floating-point. When writing, it attempts to transform the floating-point value to a native value. If this conversion fails, an `EConvertError` exception is raised.

For TJSONFloatNumber (860) this will return the native value.

For complex values, reading or writing this property will always result in an `EConvertError` exception.

See also: AsInteger (858), Value (856), AsInt64 (857), AsString (856), AsBoolean (859), IsNull (859), AsJSON (859)

25.9.23 TJSONData.AsInteger

Synopsis: Access the raw JSON value as an 32-bit integer.

Declaration: `Property AsInteger : Integer`

Visibility: public

Access: Read,Write

Description: `AsInteger` allows access to the raw value as a 32-bit integer value. When reading, it attempts to convert the native value of the data to a 32-bit integer value. When writing, it attempts to transform the 32-bit integer value to a native value. If either conversion fails, an `EConvertError` exception is raised.

For TJSONIntegerNumber (862) this will return the native value.

For complex values, reading or writing this property will always result in an `EConvertError` exception.

See also: AsFloat (858), Value (856), AsInt64 (857), AsString (856), AsBoolean (859), IsNull (859), AsJSON (859)

25.9.24 TJSONData.AsBoolean

Synopsis: Access the raw JSON value as a boolean.

Declaration: `Property AsBoolean : Boolean`

Visibility: public

Access: Read,Write

Description: `AsBoolean` allows access to the raw value as a boolean value. When reading, it attempts to convert the native value of the data to a boolean value. When writing, it attempts to transform the boolean value to a native value. For numbers this means that non-zero numbers result in `True`, a zero results in `False`. If either conversion fails, an `EConvertError` exception is raised.

For TJSONBoolean (849) this will return the native value.

For complex values, reading or writing this property will always result in an `EConvertError` exception.

See also: AsFloat (858), Value (856), AsInt64 (857), AsString (856), AsInteger (858), IsNull (859), AsJSON (859)

25.9.25 TJSONData.IsNull

Synopsis: Is the data a null value ?

Declaration: `Property IsNull : Boolean`

Visibility: public

Access: Read

Description: `IsNull` is `True` only for `JSONType=jtNull`, i.e. for a TJSONNull (863) instance. In all other cases, it is `False`. This value cannot be set.

See also: TJSONType (832), JSONType (851), TJSONNull (863), AsFloat (858), Value (856), AsInt64 (857), AsString (856), AsInteger (858), AsBoolean (859), AsJSON (859)

25.9.26 TJSONData.AsJSON

Synopsis: Return a JSON representation of the value.

Declaration: `Property AsJSON : TJSONStringType`

Visibility: public

Access: Read

Description: `AsJSON` returns a JSON representation of the value of the data. For simple values, this is just a textual representation of the object. For objects and arrays, this is an actual JSON Object or JSON Array.

See also: AsFloat (858), Value (856), AsInt64 (857), AsString (856), AsInteger (858), AsBoolean (859), AsJSON (859)

25.10 TJSONFloatNumber

25.10.1 Description

`TJSONFloatNumber` must be used whenever floating point data must be represented. It can handle TJSONFloat (830) data (normally a double). For integer data, TJSONIntegerNumber (862) or TJSONInt64Number (861) are better suited.

See also: TJSONNumber (864), TJSONFloat (830), TJSONIntegerNumber (862), TJSONInt64Number (861)

25.10.2 Method overview

Page	Method	Description
861	Clear	Clear value.
861	Clone	Clone floating point value.
860	Create	Create a new floating-point value.
860	NumberType	Kind of numerical data managed by this class.

25.10.3 TJSONFloatNumber.Create

Synopsis: Create a new floating-point value.

Declaration: `constructor Create(AValue: TJSONFloat);  Reintroduce`

Visibility: public

Description: `Create` instantiates a new JSON floating point value, and initializes it with `AValue`.

See also: TJSONIntegerNumber.Create (863), TJSONInt64Number.Create (861)

25.10.4 TJSONFloatNumber.NumberType

Synopsis: Kind of numerical data managed by this class.

Declaration: `class function NumberType : TJSONNumberType;  Override`

Visibility: public

Description: `NumberType` is overridden by `TJSONFloatNumber` to return `ntFloat`.

See also: TJSONNumberType (831), TJSONData.JSONtype (851)

25.10.5 TJSONFloatNumber.Clear

Synopsis: Clear value.

Declaration: `procedure Clear;  Override`

Visibility: public

Description: `Clear` is overridden by `TJSONFloatNumber` to set the value to 0.0

See also: TJSONData.Clear (852)

25.10.6 TJSONFloatNumber.Clone

Synopsis: Clone floating point value.

Declaration: `function Clone : TJSONData;  Override`

Visibility: public

Description: `Clone` overrides TJSONData.Clone (855) and creates an instance of the same class with the same floating-point value.

See also: TJSONData.Clone (855)

25.11 TJSONInt64Number

25.11.1 Description

`TJSONInt64Number` must be used whenever 64-bit integer data must be represented. For 32-bit integer data, TJSONIntegerNumber (862) must be used.

See also: TJSONFloatNumber (860), TJSONIntegerNumber (862)

25.11.2 Method overview

Page	Method	Description
862	Clear	Clear value.
862	Clone	Clone 64-bit integer value.
861	Create	Create a new instance of 64-bit integer JSON data.
862	NumberType	Kind of numerical data managed by this class.

25.11.3 TJSONInt64Number.Create

Synopsis: Create a new instance of 64-bit integer JSON data.

Declaration: `constructor Create(AValue: Int64);  Reintroduce`

Visibility: public

Description: `Create` instantiates a new 64-bit integer JSON data and initializes the value with `AValue`.

See also: TJSONIntegerNumber.Create (863), TJSONFloatNumber.Create (860)

25.11.4 TJSONInt64Number.NumberType

Synopsis: Kind of numerical data managed by this class.

Declaration: `class function NumberType : TJSONNumberType; Override`

Visibility: public

Description: `NumberType` is overridden by `TJSONInt64Number` to return `ntInt64`.

See also: TJSONNumberType (831), TJSONData.JSONtype (851)

25.11.5 TJSONInt64Number.Clear

Synopsis: Clear value.

Declaration: `procedure Clear; Override`

Visibility: public

Description: `Clear` is overridden by `TJSONInt64Number` to set the value to 0.

See also: TJSONData.Clear (852)

25.11.6 TJSONInt64Number.Clone

Synopsis: Clone 64-bit integer value.

Declaration: `function Clone : TJSONData; Override`

Visibility: public

Description: `Clone` overrides TJSONData.Clone (855) and creates an instance of the same class with the same 64-bit integer value.

See also: TJSONData.Clone (855)

25.12 TJSONIntegerNumber

25.12.1 Description

`TJSONIntegerNumber` must be used whenever 32-bit integer data must be represented. For 64-bit integer data, TJSONInt64Number (861) must be used.

See also: TJSONFloatNumber (860), TJSONInt64Number (861)

25.12.2 Method overview

Page	Method	Description
863	Clear	Clear value.
863	Clone	Clone 32-bit integer value.
863	Create	Create a new instance of 32-bit integer JSON data.
863	NumberType	Kind of numerical data managed by this class.

25.12.3 TJSONIntegerNumber.Create

Synopsis: Create a new instance of 32-bit integer JSON data.

Declaration: `constructor Create(AValue: Integer);  Reintroduce`

Visibility: public

Description: `Create` instantiates a new 32-bit integer JSON data and initializes the value with `AValue`.

See also: TJSONFloatNumber.Create (860), TJSONInt64Number.Create (861)

25.12.4 TJSONIntegerNumber.NumberType

Synopsis: Kind of numerical data managed by this class.

Declaration: `class function NumberType : TJSONNumberType;  Override`

Visibility: public

Description: `NumberType` is overridden by `TJSONIntegerNumber` to return `ntInteger`.

See also: TJSONNumberType (831), TJSONData.JSONtype (851)

25.12.5 TJSONIntegerNumber.Clear

Synopsis: Clear value.

Declaration: `procedure Clear;  Override`

Visibility: public

Description: `Clear` is overridden by `TJSONIntegerNumber` to set the value to 0.

See also: TJSONData.Clear (852)

25.12.6 TJSONIntegerNumber.Clone

Synopsis: Clone 32-bit integer value.

Declaration: `function Clone : TJSONData;  Override`

Visibility: public

Description: `Clone` overrides TJSONData.Clone (855) and creates an instance of the same class with the same 32-bit integer value.

See also: TJSONData.Clone (855)

25.13 TJSONNull

25.13.1 Description

`TJSONNull` must be used whenever a `null` value must be represented.

See also: TJSONFloatNumber (860), TJSONIntegerNumber (862), TJSONInt64Number (861), TJSONBoolean (849), TJSONString (877), TJSONArray (838), TJSONObject (865)

25.13.2 Method overview

Page	Method	Description
864	Clear	Clear data.
864	Clone	Clone boolean value.
864	JSONType	native JSON data type.

25.13.3 TJSONNull.JSONType

Synopsis: native JSON data type.

Declaration: `class function JSONType : TJSONtype;  Override`

Visibility: public

Description: `JSONType` is overridden by `TJSONNull` to return `jtNull`.

See also: TJSONData.JSONType (851)

25.13.4 TJSONNull.Clear

Synopsis: Clear data.

Declaration: `procedure Clear;  Override`

Visibility: public

Description: `Clear` does nothing.

See also: TJSONData.Clear (852)

25.13.5 TJSONNull.Clone

Synopsis: Clone boolean value.

Declaration: `function Clone : TJSONData;  Override`

Visibility: public

Description: `Clone` overrides TJSONData.Clone (855) and creates an instance of the same class.

See also: TJSONData.Clone (855)

25.14 TJSONNumber

25.14.1 Description

`TJSONNumber` is an abstract class which serves as the ancestor for the 3 numerical classes. It should never be instantiated directly. Instead, depending on the kind of data, one of TJSONIntegerNumber (862), TJSONInt64Number (861) or TJSONFloatNumber (860) should be instantiated.

See also: TJSONIntegerNumber (862), TJSONInt64Number (861), TJSONFloatNumber (860)

25.14.2 Method overview

Page	Method	Description
865	JSONType	native JSON data type.
865	NumberType	Kind of numerical data managed by this class.

25.14.3 TJSONNumber.JSONType

Synopsis: native JSON data type.

Declaration: `class function JSONType : TJSONtype; Override`

Visibility: public

Description: `JSONType` is overridden by `TJSONNumber` to return `jtNumber`.

See also: TJSONData.JSONType (851)

25.14.4 TJSONNumber.NumberType

Synopsis: Kind of numerical data managed by this class.

Declaration: `class function NumberType : TJSONNumberType; Virtual; Abstract`

Visibility: public

Description: `NumberType` is overridden by `TJSONNumber` descendents to return the kind of numerical data that can be managed by the class.

See also: TJSONIntegerNumber (862), TJSONInt64Number (861), TJSONFloatNumber (860), JSONType (826)

25.15 TJSONObject

25.15.1 Description

`TJSONObjectClass` is the class type of TJSONObject (865). It is used in the factory methods.

See also: TJSONObject (865), SetJSONInstanceType (836), GetJSONInstanceType (834)

25.15.2 Method overview

Page	Method	Description
870	Add	Add a name, value to the object.
870	Clear	Clear the object.
867	Clone	Clone the JSON object.
866	Create	Create a new instance of JSON object data.
870	Delete	Delete an element from the list by index.
867	Destroy	Free the JSON object.
871	Extract	Extract an element from the object.
869	Find	Find an element by name.
869	Get	Retrieve a value by name.
868	GetEnumerator	Get an object enumerator.
868	IndexOf	Return index of JSONData instance in object.
868	IndexOfName	Return index of name in item list.
868	Iterate	Iterate over all elements in the object.
867	JSONType	native JSON data type.
871	Remove	Remove item by instance.

25.15.3 Property overview

Page	Properties	Access	Description
875	Arrays	rw	Named access to JSON array values.
875	Booleans	rw	Named access to boolean values.
872	Elements	rw	Name-based access to JSON values in the object.
873	Floats	rw	Named access to float values.
873	Int64s	rw	Named access to int64 values.
873	Integers	rw	Named access to integer values.
874	LargeInts	rw	Get or set elements as LargeInt values.
872	Names	r	Indexed access to the names of elements.
873	Nulls	rw	Named access to `null` values.
876	Objects	rw	Named access to JSON object values.
874	QWords	rw	Named access to QWord values.
875	Strings	rw	Named access to string values.
872	Types	r	Types of values in the object.
874	UnicodeStrings	rw	Named access to Unicode string values.
871	UnquotedMemberNames	rw	Should member names be written unquoted or quoted in JSON.

25.15.4 TJSONObject.Create

Synopsis: Create a new instance of JSON object data.

Declaration:
```
constructor Create;  Reintroduce
constructor Create(const Elements: Array of const);  Overload
```

Visibility: public

Description: `Create` creates a new JSON object instance, and initializes the data with `Elements`. Elements is an array containing an even number of items, alternating a name and a value. The names must be strings, and the values are converted to various TJSONData (850) instances. If a value is an instance of `TJSONData`, it is added to the object array as-is.

The data type of the inserted objects is determined from the type of data passed to it, with a natural mapping. A `Nil` pointer will be inserted as a `TJSONNull` value. The following gives an example:

```
Var
  O : TJSONObject;

begin
  O:=TJSONObject.Create(['Age',44,
                         'Firstname','Michael',
                         'Lastname','Van Canneyt']);
```

Errors: An `EConvertError` exception is raised in one of the following cases:

1.If an odd number of arguments is passed

2.an item where a name is expected does not contain a string

3.A value contains an invalid class

4.A value of a not recognized data type (pointer) is inserted in the elements

See also: Add (870), GetJSONInstanceType (834)

25.15.5 TJSONObject.Destroy

Synopsis: Free the JSON object.

Declaration: `destructor Destroy;  Override`

Visibility: public

Description: `Destroy` will delete all elements in the array and clean up the TJSONObject (865) instance.

See also: TJSONObject.Clear (870), TJSONObject.Create (866)

25.15.6 TJSONObject.JSONType

Synopsis: native JSON data type.

Declaration: `class function JSONType : TJSONtype;  Override`

Visibility: public

Description: `JSONType` is overridden by `TJSONObject` to return `jtObject`.

See also: TJSONData.JSONType (851)

25.15.7 TJSONObject.Clone

Synopsis: Clone the JSON object.

Declaration: `function Clone : TJSONData;  Override`

Visibility: public

Description: `Clone` creates a new `TJSONObject`, clones all elements in the array and adds them to the newly created array with the same names as they were in the array.

See also: TJSONData.Clone (855)

25.15.8 TJSONObject.GetEnumerator

Synopsis: Get an object enumerator.

Declaration: `function GetEnumerator : TBaseJSONEnumerator;  Override`

Visibility: public

Description: `GetEnumerator` is overridden in `TJSONObject` so it returns an object enumerator. The array enumerator will return all the elements in the array, and stores their name in the `Key` and index in the `KeyNum` members of TJSONEnum (830).

See also: TJSONEnum (830), TJSONData.GetEnumerator (852)

25.15.9 TJSONObject.Iterate

Synopsis: Iterate over all elements in the object.

Declaration: `procedure Iterate(Iterator: TJSONObjectIterator; Data: TObject)`

Visibility: public

Description: `Iterate` iterates over all elements in the object, passing them one by one with name and value to the `Iterator` callback, together with the `Data` parameter. The iteration stops when all elements have been passed or when the iterator callback returned `False` in the `Continue` parameter.

See also: TJSONObjectIterator (832)

25.15.10 TJSONObject.IndexOf

Synopsis: Return index of JSONData instance in object.

Declaration: `function IndexOf(Item: TJSONData) : Integer`

Visibility: public

Description: `IndexOf` compares all elements in the object with `Obj` and returns the index (in the TJSONData.Items (856) property) of the element instance that equals `Obj`. The actual instances are compared, not the JSON value. If none of the elements match, the function returns -1.

See also: Clear (870), IndexOfName (868)

25.15.11 TJSONObject.IndexOfName

Synopsis: Return index of name in item list.

Declaration:
```
function IndexOfName(const AName: TJSONStringType;
                     CaseInsensitive: Boolean) : Integer
```

Visibility: public

Description: `IndexOfName` compares the names of all elements in the object with `AName` and returns the index (in the TJSONData.Items (856) property) of the element instance whose name matched `AName`. If none of the element's names match, the function returns -1.

Since JSON is a case-sensitive specification, the names are searched case-sensitively by default. This can be changed by setting the optional `CaseInsensitive` parameter to `True`

See also: IndexOf (868)

25.15.12 TJSONObject.Find

Synopsis: Find an element by name.

Declaration:
```
function Find(const AName: TJSONStringType) : TJSONData;  Overload
function Find(const AName: TJSONStringType; AType: TJSONtype)
              : TJSONData;  Overload
function Find(const key: TJSONStringType; out AValue: TJSONData)
              : Boolean
function Find(const key: TJSONStringType; out AValue: TJSONObject)
              : Boolean
function Find(const key: TJSONStringType; out AValue: TJSONArray)
              : Boolean
function Find(const key: TJSONStringType; out AValue: TJSONString)
              : Boolean
function Find(const key: TJSONStringType; out AValue: TJSONBoolean)
              : Boolean
function Find(const key: TJSONStringType; out AValue: TJSONNumber)
              : Boolean
```

Visibility: public

Description: `Find` compares the names of all elements in the object with `AName` and returns the matching element. If none of the element's names match, the function returns `Nil`

Since JSON is a case-sensitive specification, the names are searched case-sensitively.

If `AType` is specified then the element's type must also match the specified type.

See also: IndexOf (868), IndexOfName (868)

25.15.13 TJSONObject.Get

Synopsis: Retrieve a value by name.

Declaration:
```
function Get(const AName: string) : TJSONVariant
function Get(const AName: string; ADefault: TJSONFloat) : TJSONFloat
function Get(const AName: string; ADefault: Integer) : Integer
function Get(const AName: string; ADefault: Int64) : Int64
function Get(const AName: string; ADefault: QWord) : QWord
function Get(const AName: string; ADefault: TJSONUnicodeStringType)
             : TJSONUnicodeStringType
function Get(const AName: string; ADefault: Boolean) : Boolean
function Get(const AName: string; ADefault: TJSONStringType)
             : TJSONStringType
function Get(const AName: string; ADefault: TJSONArray) : TJSONArray
function Get(const AName: string; ADefault: TJSONObject) : TJSONObject
```

Visibility: public

Description: `Get` can be used to retrieve a value by name. If an element with name equal to `AName` exists, and its type corresponds to the type of the `ADefault`, then the value is returned. If no element element with the correct type exists, the `ADefault` value is returned.

If no default is specified, the value is returned as a variant type, or Null if no value was found.

The other value retrieval properties such as Integers (873), Int64s (873), Booleans (875), Strings (875), Floats (873), Arrays (875), and Objects (876) will raise an exception if the name is not found. The `Get` function does not raise an exception.

See also: Integers (873), Int64s (873), Booleans (875), Strings (875), Floats (873), Arrays (875), Objects (876)

25.15.14 TJSONObject.Clear

Synopsis: Clear the object.

Declaration:
```
procedure Clear;  Override
```

Visibility: public

Description: `Clear` clears the object and frees all elements in it. After the call to `Clear`, Count (856) returns 0.

See also: Delete (870), Extract (871)

25.15.15 TJSONObject.Add

Synopsis: Add a name, value to the object.

Declaration:
```
function Add(const AName: TJSONStringType; AValue: TJSONData) : Integer
              ;  Overload
function Add(const AName: TJSONStringType; AValue: Boolean) : Integer
              ;  Overload
function Add(const AName: TJSONStringType; AValue: TJSONFloat) : Integer
              ;  Overload
function Add(const AName: TJSONStringType;
             const AValue: TJSONStringType) : Integer;  Overload
function Add(const AName: string; AValue: TJSONUnicodeStringType)
              : Integer;  Overload
function Add(const AName: TJSONStringType; Avalue: Int64) : Integer
              ;  Overload
function Add(const AName: TJSONStringType; Avalue: QWord) : Integer
              ;  Overload
function Add(const AName: TJSONStringType; Avalue: Integer) : Integer
              ;  Overload
function Add(const AName: TJSONStringType) : Integer;  Overload
function Add(const AName: TJSONStringType; AValue: TJSONArray) : Integer
              ;  Overload
```

Visibility: public

Description: `Add` adds the value `AValue` with name `AName` to the object. If the value is not a TJSONData (850) descendent, then it is converted to a `TJSONData` value, and it returns the `TJSONData` descendent that was created to add the value.

The properties Integers (873), Int64s (873), Booleans (875), Strings (875), Floats (873), Arrays (875) and Objects (876) will not raise an exception if an existing name is used. They will overwrite any existing value.

Errors: If a value with the same name already exists, an exception is raised.

See also: Integers (873), Int64s (873), Booleans (875), Strings (875), Floats (873), Arrays (875), Objects (876)

25.15.16 TJSONObject.Delete

Synopsis: Delete an element from the list by index.

Declaration:
```
procedure Delete(Index: Integer)
procedure Delete(const AName: string)
```

Visibility: public

Description: `Delete` deletes the element with given `Index` or `AName` from the list. The TJSONData (850) element is freed. If a non-existing name is specified, no value is deleted.

Errors: If an invalid index is passed, an exception is raised.

See also: Clear (870), Add (870), Extract (871), Exchange (865)

25.15.17 TJSONObject.Remove

Synopsis: Remove item by instance.

Declaration:
```
procedure Remove(Item: TJSONData)
```

Visibility: public

Description: `Remove` will locate the value `Item` in the list of values, and removes it if it exists. The item is freed.

See also: Delete (870), Extract (871)

25.15.18 TJSONObject.Extract

Synopsis: Extract an element from the object.

Declaration:
```
function Extract(Index: Integer) : TJSONData
function Extract(const AName: string) : TJSONData
```

Visibility: public

Description: `Extract` removes the element at position `Index` or with the `AName` from the list, just as Delete (826) does. In difference with `Delete`, it does not free the object instance. Instead, it returns the extracted element. The result is `Nil` if a non-existing name is specified.

See also: Delete (826), Clear (826), Insert (826), Add (826)

25.15.19 TJSONObject.UnquotedMemberNames

Synopsis: Should member names be written unquoted or quoted in JSON.

Declaration:
```
Property UnquotedMemberNames : Boolean
```

Visibility: public

Access: Read,Write

Description: `UnquotedMemberNames` can be set to let `AsJSON` write the member names of a JSON object without quotes (`True`) or with quotes (`False`) around the member names.

When the value is `False`, JSON is written as:

```
{
  "name" : "Free Pascal",
  "type" : "Compiler"
}
```

When the value is `True`, JSON is written as:

```
{
  name : "Free Pascal",
  type : "Compiler"
}
```

Care must be taken when setting this property: The JSON standard requires the quotes to be written, but since JSON is mostly consumed in a Javascipt engine, the unquoted values are usually also accepted.

See also: TJSONData.CompressedJSON (855)

25.15.20 TJSONObject.Names

Synopsis: Indexed access to the names of elements.

Declaration: `Property Names[Index: Integer]: TJSONStringType`

Visibility: public

Access: Read

Description: `Names` allows to retrieve the names of the elements in the object. The index is zero-based, running from 0 to `Count-1`.

See also: Types (872), Elements (872)

25.15.21 TJSONObject.Elements

Synopsis: Name-based access to JSON values in the object.

Declaration: `Property Elements[AName: string]: TJSONData; default`

Visibility: public

Access: Read,Write

Description: `Elements` allows to retrieve the JSON values of the elements in the object by name. If a non-existent name is specified, an EJSON (837) exception is raised.

See also: Items (856), Names (872), Types (872), Integers (873), Int64s (873), Booleans (875), Strings (875), Floats (873), Arrays (875), Objects (876)

25.15.22 TJSONObject.Types

Synopsis: Types of values in the object.

Declaration: `Property Types[AName: string]: TJSONtype`

Visibility: public

Access: Read

Description: `Types` allows to retrieve the JSON types of the elements in the object by name. If a non-existent name is specified, an EJSON (837) exception is raised.

See also: Items (856), Names (872), Elements (872), Integers (873), Int64s (873), Booleans (875), Strings (875), Floats (873), Arrays (875), Nulls (873), Objects (876)

25.15.23 TJSONObject.Nulls

Synopsis: Named access to `null` values.

Declaration: `Property Nulls[AName: string]: Boolean`

Visibility: public

Access: Read,Write

Description: `Nulls` allows to retrieve or set the NULL values in the object by name. If a non-existent name is specified, an EJSON (837) exception is raised when reading. When writing, any existing value is replaced by a `null` value.

See also: TJSONData.Items (856), TJSONObject.Names (872), Elements (826), Integers (826), Int64s (826), Booleans (826), Strings (826), Floats (826), Arrays (826), Types (826), Objects (826)

25.15.24 TJSONObject.Floats

Synopsis: Named access to float values.

Declaration: `Property Floats[AName: string]: TJSONFloat`

Visibility: public

Access: Read,Write

Description: `Floats` allows to retrieve or set the float values in the object by name. If a non-existent name is specified, an EJSON (837) exception is raised when reading. When writing, any existing value is replaced by the specified floating-point value.

See also: Items (856), Names (872), Elements (872), Integers (873), Int64s (873), Booleans (875), Strings (875), Nulls (873), Arrays (875), Types (872), Objects (876)

25.15.25 TJSONObject.Integers

Synopsis: Named access to integer values.

Declaration: `Property Integers[AName: string]: Integer`

Visibility: public

Access: Read,Write

Description: `Integers` allows to retrieve or set the integer values in the object by name. If a non-existent name is specified, an EJSON (837) exception is raised when reading. When writing, any existing value is replaced by the specified integer value.

See also: TJSONData.Items (856), Names (872), Elements (872), Floats (873), Int64s (873), Booleans (875), Strings (875), Nulls (873), Arrays (875), Types (872), Objects (876)

25.15.26 TJSONObject.Int64s

Synopsis: Named access to int64 values.

Declaration: `Property Int64s[AName: string]: Int64`

Visibility: public

Access: Read,Write

Description: `Int64s` allows to retrieve or set the int64 values in the object by name. If a non-existent name is specified, an EJSON (837) exception is raised when reading. When writing, any existing value is replaced by the specified int64 value.

See also: Items (856), Names (872), Elements (872), Floats (873), Integers (873), Booleans (875), Strings (875), Nulls (873), Arrays (875), Types (872), Objects (876)

25.15.27 TJSONObject.QWords

Synopsis: Named access to QWord values.

Declaration: `Property QWords[AName: string]: QWord`

Visibility: public

Access: Read,Write

Description: `QWords` allows to retrieve or set the QWord values in the object by name. If a non-existent name is specified, an EJSON (837) exception is raised when reading. When writing, any existing value is replaced by the specified QWord value.

See also: TJSONData.Items (856), Names (872), Elements (872), Floats (873), Integers (873), Booleans (875), Strings (875), Nulls (873), Arrays (875), Types (872), Objects (876)

25.15.28 TJSONObject.LargeInts

Synopsis: Get or set elements as LargeInt values.

Declaration: `Property LargeInts[AName: string]: TJSONLargeInt`

Visibility: public

Access: Read,Write

Description: `LargeInts` gives direct access to the TJSONData.AsLargeInt (858) property when reading. Reading it is the equivalent to accessing

```
Items[Index].AsLargeInt
```

When writing, it will check if an Largeint integer JSON value is located at the given location, and replace it with the new value. If a non-large-integer JSON value is there, it is replaced with the written value.

See also: TJSONData.AsLargeInt (858), TJSONData.Items (856), TJSONData.IsNull (859), TJSONObject.Types (872), TJSONObject.Integers (873), TJSONObject.Floats (873), TJSONObject.Strings (875), TJSONObject.Booleans (875), TJSONObject.QWords (874)

25.15.29 TJSONObject.UnicodeStrings

Synopsis: Named access to Unicode string values.

Declaration: `Property UnicodeStrings[AName: string]: TJSONUnicodeStringType`

Visibility: public

Access: Read,Write

Description: `Strings` allows to retrieve or set the Unicode string values in the object by name. If a non-existent name is specified, an EJSON (837) exception is raised when reading. When writing, any existing value is replaced by the specified UnicodeString value.

See also: Items (856), Names (872), Elements (872), Floats (873), Integers (873), Booleans (875), Int64s (873), Nulls (873), Arrays (875), Types (872), Objects (876), Strings (875)

25.15.30 TJSONObject.Strings

Synopsis: Named access to string values.

Declaration: `Property Strings[AName: string]: TJSONStringType`

Visibility: public

Access: Read,Write

Description: `Strings` allows to retrieve or set the string values in the object by name. If a non-existent name is specified, an EJSON (837) exception is raised when reading. When writing, any existing value is replaced by the specified string value.

See also: Items (856), Names (872), Elements (872), Floats (873), Integers (873), Booleans (875), Int64s (873), Nulls (873), Arrays (875), Types (872), Objects (876), UnicodeStrings (874)

25.15.31 TJSONObject.Booleans

Synopsis: Named access to boolean values.

Declaration: `Property Booleans[AName: string]: Boolean`

Visibility: public

Access: Read,Write

Description: `Booleans` allows to retrieve or set the boolean values in the object by name. If a non-existent name is specified, an EJSON (837) exception is raised when reading. When writing, any existing value is replaced by the specified boolean value.

See also: Items (856), Names (872), Elements (872), Floats (873), Integers (873), Strings (875), Int64s (873), Nulls (873), Arrays (875), Types (872), Objects (876)

25.15.32 TJSONObject.Arrays

Synopsis: Named access to JSON array values.

Declaration: `Property Arrays[AName: string]: TJSONArray`

Visibility: public

Access: Read,Write

Description: `Arrays` allows to retrieve or set the JSON array values in the object by name. If a non-existent name is specified, an EJSON (837) exception is raised when reading. When writing, any existing value is replaced by the specified JSON array.

See also: Items (856), Names (872), Elements (872), Floats (873), Integers (873), Strings (875), Int64s (873), Nulls (873), Booleans (875), Types (872), Objects (876)

25.15.33 TJSONObject.Objects

Synopsis: Named access to JSON object values.

Declaration: `Property Objects[AName: string]: TJSONObject`

Visibility: public

Access: Read,Write

Description: `Objects` allows to retrieve or set the JSON object values in the object by name. If a non-existent name is specified, an EJSON (837) exception is raised when reading. When writing, any existing value is replaced by the specified JSON object.

See also: TJSONData.Items (856), Names (872), Elements (872), Floats (873), Integers (873), Strings (875), Int64s (873), Nulls (873), Booleans (875), Types (872), Arrays (875)

25.16 TJSONQWordNumber

25.16.1 Description

`TJSONQWordNumber` must be used whenever 64-bit unsigned integer data must be represented. For 32-bit integer data, TJSONIntegerNumber (862) must be used.For 64-bit signed integer data, TJSONInt64Number (861) must be used.

See also: TJSONFloatNumber (860), TJSONIntegerNumber (862), TJSONInt64Number (861)

25.16.2 Method overview

Page	Method	Description
877	Clear	Clear value.
877	Clone	Clone 64-bit unsigned integer value.
876	Create	Create a new instance of 64-bit unsigned integer JSON data.
876	NumberType	Kind of numerical data managed by this class.

25.16.3 TJSONQWordNumber.Create

Synopsis: Create a new instance of 64-bit unsigned integer JSON data.

Declaration: `constructor Create(AValue: QWord);  Reintroduce`

Visibility: public

Description: `Create` instantiates a new 64-bit unsigned integer JSON data and initializes the value with `AValue`.

See also: TJSONIntegerNumber.Create (863), TJSONInt64Number.Create (861), TJSONFloatNumber.Create (860)

25.16.4 TJSONQWordNumber.NumberType

Synopsis: Kind of numerical data managed by this class.

Declaration: `class function NumberType : TJSONNumberType;  Override`

Visibility: public

Description: `NumberType` is overridden by `TJSONQwordNumber` to return `ntQWord`.

See also: TJSONNumberType (831), TJSONData.JSONtype (851)

25.16.5 TJSONQWordNumber.Clear

Synopsis: Clear value.

Declaration: `procedure Clear;  Override`

Visibility: public

Description: `Clear` is overridden by `TJSONQwordNumber` to set the value to 0.

See also: TJSONData.Clear (852)

25.16.6 TJSONQWordNumber.Clone

Synopsis: Clone 64-bit unsigned integer value.

Declaration: `function Clone : TJSONData;  Override`

Visibility: public

Description: `Clone` overrides TJSONData.Clone (855) and creates an instance of the same class with the 64-bit unsigned integer value.

See also: TJSONData.Clone (855)

25.17 TJSONString

25.17.1 Description

`TJSONString` must be used whenever string data must be represented. Currently the implementation uses an ANSI string to hold the data. This means that to correctly hold Unicode data, a UTF-8 encoding must be used.

See also: TJSONFloatNumber (860), TJSONIntegerNumber (862), TJSONInt64Number (861), TJSONBoolean (849), TJSONNull (863), TJSONArray (838), TJSONObject (865)

25.17.2 Method overview

Page	Method	Description
878	Clear	Clear value.
878	Clone	Clone string value.
877	Create	Create a new instance of string JSON data.
878	JSONType	native JSON data type.

25.17.3 TJSONString.Create

Synopsis: Create a new instance of string JSON data.

Declaration:
```
constructor Create(const AValue: TJSONStringType);  Reintroduce
constructor Create(const AValue: TJSONUnicodeStringType);  Reintroduce
```

Visibility: public

Description: `Create` instantiates a new string JSON data and initializes the value with `AValue`. Currently the implementation uses an ANSI string to hold the data. This means that to correctly hold Unicode data, a UTF-8 encoding must be used.

See also: TJSONIntegerNumber.Create (863), TJSONFloatNumber.Create (860), TJSONInt64Number.Create (861), TJSONBoolean.Create (849), TJSONArray.Create (839), TJSONObject.Create (866)

25.17.4 TJSONString.JSONType

Synopsis: native JSON data type.

Declaration: `class function JSONType : TJSONtype;  Override`

Visibility: public

Description: `JSONType` is overridden by `TJSONString` to return `jtString`.

See also: TJSONData.JSONType (851)

25.17.5 TJSONString.Clear

Synopsis: Clear value.

Declaration: `procedure Clear;  Override`

Visibility: public

Description: `Clear` is overridden by `TJSONString` to set the value to the empty string ''.

See also: TJSONData.Clear (852)

25.17.6 TJSONString.Clone

Synopsis: Clone string value.

Declaration: `function Clone : TJSONData;  Override`

Visibility: public

Description: `Clone` overrides TJSONData.Clone (855) and creates an instance of the same class with the same string value.

See also: TJSONData.Clone (855)

Chapter 26

Reference for unit 'fpjsonapply'

26.1 Used units

Table 26.1: Used units by unit 'fpjsonapply'

Name	Page
Classes	??
fpjson	826
System	??
sysutils	??

26.2 Overview

This unit provides the TJSONApplier (880) component class for merging JSON objects. It allows applying (merging) elements from one JSON object to another with various configuration options for ownership control, case sensitivity, type handling, and output formatting.

26.3 Constants, types and variables

26.3.1 Types

```
TOwnsJSON = (ojSource,ojApply)
```

Table 26.2: Enumeration values for type TOwnsJSON

Value	Explanation
ojApply	The apply JSON object is owned by the applier component
ojSource	The source JSON object is owned by the applier component

The `TOwnsJSON` enumeration defines which JSON objects are owned by the TJSONApplier (880) component. When a JSON object is owned, it will be automatically freed when the component is destroyed or when a new object is assigned.

ojSource The source JSON object is owned by the applier component

ojApply The apply JSON object is owned by the applier component

```
TOwnsJSONs = Set of TOwnsJSON
```

The `TOwnsJSONs` type is a set of TOwnsJSON (879) enumeration values. It specifies which JSON objects are owned and managed by the TJSONApplier (880) component. Owned objects will be automatically freed when no longer needed.

26.4 TJSONApplier

26.4.1 Description

The `TJSONApplier` class is a component that provides functionality to apply (merge) elements from one JSON object to another. It supports loading JSON objects from files or working with existing objects in memory, with various configuration options for controlling the merge behavior. The component can merge at specific paths within JSON structures, handle type conflicts, control case sensitivity, and manage object ownership.

See also: TOwnsJSON (879), TOwnsJSONs (880)

26.4.2 Method overview

Page	Method	Description
881	destroy	Destructor that frees owned JSON objects and cleans up resources
881	Execute	Executes the JSON merge operation from apply JSON to source JSON

26.4.3 Property overview

Page	Properties	Access	Description
884	ApplyFileName	rw	Filename to load apply JSON from if ApplyJSON is not set
882	ApplyJSON	rw	JSON object containing elements to apply to the source
884	ApplyPath	rw	JSON path in apply object where merging data should be taken from
885	CaseInsensitive	rw	Controls whether property name matching is case sensitive
884	CloneSource	rw	Controls whether to make a clone copy of source JSON before applying changes
884	DestFileName	rw	Filename to write the destination JSON to after merging
882	DestJSON	r	Destination JSON object containing the merge result
885	ForceCorrectType	rw	Controls whether to overwrite properties when types differ between source and apply JSON
886	Formatted	rw	Controls whether output JSON is formatted with indentation and line breaks
883	OwnsJSON	rw	Set specifying which JSON objects are owned by the component
885	RemoveNonExisting	rw	Controls whether to remove properties from source that do not exist in apply JSON
883	SourceFileName	rw	Filename to load source JSON from if SourceJSON is not set
882	SourceJSON	rw	Source JSON object to merge into
883	SourcePath	rw	JSON path in source object where merging should start

26.4.4 TJSONApplier.destroy

Synopsis: Destructor that frees owned JSON objects and cleans up resources

Declaration: `destructor destroy;  Override`

Visibility: public

Description: The `destroy` destructor frees any JSON objects that are owned by the component as indicated by the OwnsJSON (883) property. It also frees the destination JSON object if it differs from the source JSON object.

See also: TJSONApplier.OwnsJSON (883), TJSONApplier.SourceJSON (882), TJSONApplier.ApplyJSON (882), TJSONApplier.DestJSON (882)

26.4.5 TJSONApplier.Execute

Synopsis: Executes the JSON merge operation from apply JSON to source JSON

Declaration: `procedure Execute;  Virtual`

Visibility: public

Description: The `Execute` method performs the main JSON merge operation. It loads the source and apply JSON objects from files if they are not already set, then applies the elements from the apply JSON object to the source JSON object. The result is stored in the DestJSON (882) property. If file paths are specified, it starts merging at those specific locations within the JSON structures. If a destination filename is specified, the result is automatically saved to that file.

Errors: Raises `EJSON` exceptions when:

- Source JSON object is empty
- Apply JSON object is empty
- JSON files do not contain valid JSON objects
- Specified source or apply paths are not found or do not point to JSON objects

See also: TJSONApplier.SourceJSON (882), TJSONApplier.ApplyJSON (882), TJSONApplier.DestJSON (882), TJSONApplier.SourcePath (883), TJSONApplier.ApplyPath (884)

26.4.6 TJSONApplier.SourceJSON

Synopsis: Source JSON object to merge into

Declaration: `Property SourceJSON : TJSONObject`

Visibility: public

Access: Read,Write

Description: The `SourceJSON` property holds the source JSON object that serves as the base for the merge operation. If not set directly, it will be loaded automatically from the file specified in SourceFileName (883) when Execute (881) is called. When a new value is assigned, any previously owned source JSON object is freed.

See also: TJSONApplier.SourceFileName (883), TJSONApplier.DestJSON (882), TJSONApplier.OwnsJSON (883)

26.4.7 TJSONApplier.ApplyJSON

Synopsis: JSON object containing elements to apply to the source

Declaration: `Property ApplyJSON : TJSONObject`

Visibility: public

Access: Read,Write

Description: The `ApplyJSON` property holds the JSON object whose elements will be merged into the source JSON object. If not set directly, it will be loaded automatically from the file specified in ApplyFileName (884) when Execute (881) is called. When a new value is assigned, any previously owned apply JSON object is freed.

See also: TJSONApplier.ApplyFileName (884), TJSONApplier.SourceJSON (882), TJSONApplier.OwnsJSON (883)

26.4.8 TJSONApplier.DestJSON

Synopsis: Destination JSON object containing the merge result

Declaration: `Property DestJSON : TJSONObject`

Visibility: public

Access: Read

Description: The `DestJSON` property contains the result of the merge operation after Execute (881) is called. If CloneSource (884) is `True`, this will be a clone of the source JSON object. If CloneSource (884) is `False`, this will be the same reference as the source JSON object. This property is read-only and is set automatically during the merge operation.

See also: TJSONApplier.SourceJSON (882), TJSONApplier.CloneSource (884), TJSONApplier.Execute (881)

26.4.9 TJSONApplier.OwnsJSON

Synopsis: Set specifying which JSON objects are owned by the component

Declaration: `Property OwnsJSON : TOwnsJSONs`

Visibility: public

Access: Read,Write

Description: The `OwnsJSON` property is a set of TOwnsJSON (879) values that specifies which JSON objects are owned and managed by the component. Owned objects are automatically freed when the component is destroyed or when new objects are assigned. This property is automatically updated when JSON objects are loaded from files.

See also: TOwnsJSON (879), TOwnsJSONs (880), TJSONApplier.SourceJSON (882), TJSONApplier.ApplyJSON (882)

26.4.10 TJSONApplier.SourceFileName

Synopsis: Filename to load source JSON from if SourceJSON is not set

Declaration: `Property SourceFileName : string`

Visibility: published

Access: Read,Write

Description: The `SourceFileName` property specifies the file path from which to load the source JSON object. If SourceJSON (882) is not set when Execute (881) is called, the JSON object will be loaded from this file. When loaded from file, the component will take ownership of the JSON object.

See also: TJSONApplier.SourceJSON (882), TJSONApplier.OwnsJSON (883), TJSONApplier.Execute (881)

26.4.11 TJSONApplier.SourcePath

Synopsis: JSON path in source object where merging should start

Declaration: `Property SourcePath : string`

Visibility: published

Access: Read,Write

Description: The `SourcePath` property specifies a JSON path within the source JSON object where the merge operation should begin. If empty, merging starts at the root of the source object. The path must point to an existing JSON object within the source structure. JSON paths use dot notation to navigate nested objects.

See also: TJSONApplier.SourceJSON (882), TJSONApplier.ApplyPath (884), TJSONApplier.Execute (881)

26.4.12 TJSONApplier.ApplyFileName

Synopsis: Filename to load apply JSON from if ApplyJSON is not set

Declaration: `Property ApplyFileName : string`

Visibility: published

Access: Read,Write

Description: The `ApplyFileName` property specifies the file path from which to load the apply JSON object. If ApplyJSON (882) is not set when Execute (881) is called, the JSON object will be loaded from this file. When loaded from file, the component will take ownership of the JSON object.

See also: TJSONApplier.ApplyJSON (882), TJSONApplier.OwnsJSON (883), TJSONApplier.Execute (881)

26.4.13 TJSONApplier.ApplyPath

Synopsis: JSON path in apply object where merging data should be taken from

Declaration: `Property ApplyPath : string`

Visibility: published

Access: Read,Write

Description: The `ApplyPath` property specifies a JSON path within the apply JSON object from which elements should be merged. If empty, elements are taken from the root of the apply object. The path must point to an existing JSON object within the apply structure. JSON paths use dot notation to navigate nested objects.

See also: TJSONApplier.ApplyJSON (882), TJSONApplier.SourcePath (883), TJSONApplier.Execute (881)

26.4.14 TJSONApplier.DestFileName

Synopsis: Filename to write the destination JSON to after merging

Declaration: `Property DestFileName : string`

Visibility: published

Access: Read,Write

Description: The `DestFileName` property specifies the file path where the result of the merge operation should be saved. If this property is set, the destination JSON object will be automatically saved to this file after Execute (881) completes. If empty, no automatic saving is performed.

See also: TJSONApplier.DestJSON (882), TJSONApplier.Execute (881), TJSONApplier.Formatted (886)

26.4.15 TJSONApplier.CloneSource

Synopsis: Controls whether to make a clone copy of source JSON before applying changes

Declaration: `Property CloneSource : Boolean`

Visibility: published

Access: Read,Write

Description: The `CloneSource` property determines whether a clone copy of the source JSON object is made before applying changes. When `True`, the source JSON object is cloned and changes are applied to the clone, leaving the original unchanged. When `False`, changes are applied directly to the source JSON object. The default value is `False`.

See also: TJSONApplier.SourceJSON (882), TJSONApplier.DestJSON (882), TJSONApplier.Execute (881)

26.4.16 TJSONApplier.CaseInsensitive

Synopsis: Controls whether property name matching is case sensitive

Declaration: `Property CaseInsensitive : Boolean`

Visibility: published

Access: Read,Write

Description: The `CaseInsensitive` property controls whether property names are matched case-sensitively during the merge operation. When `True`, property names are compared ignoring case differences. When `False`, property names must match exactly including case. The default value is `False`.

See also: TJSONApplier.Execute (881), TJSONApplier.RemoveNonExisting (885)

26.4.17 TJSONApplier.ForceCorrectType

Synopsis: Controls whether to overwrite properties when types differ between source and apply JSON

Declaration: `Property ForceCorrectType : Boolean`

Visibility: published

Access: Read,Write

Description: The `ForceCorrectType` property determines the behavior when a property exists in both source and apply JSON objects but with different data types. When `True`, the property in the source is deleted and replaced with the property from the apply JSON. When `False`, properties with different types are handled according to their specific types (objects are merged recursively). The default value is `False`.

See also: TJSONApplier.Execute (881), TJSONApplier.CaseInsensitive (885)

26.4.18 TJSONApplier.RemoveNonExisting

Synopsis: Controls whether to remove properties from source that do not exist in apply JSON

Declaration: `Property RemoveNonExisting : Boolean`

Visibility: published

Access: Read,Write

Description: The `RemoveNonExisting` property determines whether properties that exist in the source JSON but not in the apply JSON should be removed after merging. When `True`, properties in the source that are not present in the apply JSON are deleted. When `False`, existing properties in the source are preserved even if they do not exist in the apply JSON. The default value is `False`.

See also: TJSONApplier.Execute (881), TJSONApplier.CaseInsensitive (885)

26.4.19 TJSONApplier.Formatted

Synopsis: Controls whether output JSON is formatted with indentation and line breaks

Declaration: `Property Formatted : Boolean`

Visibility: published

Access: Read,Write

Description: The `Formatted` property controls whether the JSON output is formatted with indentation and line breaks for better readability. When `True`, the output JSON is formatted with proper indentation. When `False`, the output JSON is written as a compact string without formatting. This property affects the output when writing to the DestFileName (884). The default value is `False`.

See also: TJSONApplier.DestFileName (884), TJSONApplier.Execute (881)

Chapter 27

Reference for unit 'fpjsontopas'

27.1 Used units

Table 27.1: Used units by unit 'fpjsontopas'

Name	Page
Classes	??
fpjson	826
jsonparser	1321
System	??
sysutils	??

27.2 Overview

The `fpJsonToPas` unit provides functionality to generate Object Pascal class definitions from JSON data structures. The main class TJSONToPascal (889) analyzes JSON input and creates complete Pascal unit code with classes that can serialize to and from JSON.

The generated Pascal classes include properties for each JSON field, optional load/save methods for JSON serialization, and proper type mappings from JSON types to Pascal types.

27.3 Using fpjsontopas to generate Pascal classes from JSON

The `fpjsontopas` unit allows you to generate Object Pascal class definitions from JSON data. This is useful for creating strongly-typed classes that can serialize to and from JSON format.

The basic workflow involves:

1. Create a TJSONToPascal (889) instance
2. Set the JSON data using the JSON (892) property
3. Configure options like DestUnitName (893) and ToplevelObjectClassName (894)
4. Call Execute (891) to generate the Pascal code
5. Retrieve the generated code from the Code (892) property

Here's a simple example that generates Pascal classes from JSON data:

```
program JSONToPascalDemo;

{$mode objfpc}{$H+}

uses
  Classes, SysUtils, fpjsontopas;

var
  Generator: TJSONToPascal;
  JSONData: string;
  i: Integer;
begin
  JSONData := '{"name": "John Doe", "age": 30, "active": true, "address": {"street":

  Generator := TJSONToPascal.Create(nil);
  try
    // Configure the generator
    Generator.JSON := JSONData;
    Generator.DestUnitName := 'PersonUnit';
    Generator.ToplevelObjectClassName := 'TPerson';
    Generator.Options := [jpoGenerateLoad, jpoGenerateSave];

    // Generate the Pascal code
    Generator.Execute;

    // Output the generated code
    Writeln('Generated Pascal unit:');
    Writeln;
    for i := 0 to Generator.Code.Count - 1 do
      Writeln(Generator.Code[i]);

  finally
    Generator.Free;
  end;
end.
```

This example will generate a complete Pascal unit with `TPerson` and `TPersonAddress` classes, including properties for each JSON field and methods to load from/save to JSON.

You can customize the generation process using various properties:

- DefaultParentName (893): Base class for generated objects
- FieldPrefix (894): Prefix for private field names
- IndentSize (893): Code indentation size
- ExtraUnitNames (893): Additional units in the uses clause
- PropertyMap (892): Manual type mappings for specific JSON paths

27.4 Constants, types and variables

27.4.1 Types

```
TJSONToPascalOption = (jpoUseSetter,jpoGenerateLoad,
  jpoUnknownLoadPropsError,jpoDelphiJSON,
  jpoLoadCaseInsensitive,jpoGenerateSave)
```

Table 27.2: Enumeration values for type TJSONToPascalOption

Value	Explanation
jpoDelphiJSON	Generate Delphi-compatible JSON code
jpoGenerateLoad	Generate LoadFromJSON methods
jpoGenerateSave	Generate SaveToJSON methods
jpoLoadCaseInsensitive	Make property loading case-insensitive
jpoUnknownLoadPropsError	Raise error for unknown properties during loading
jpoUseSetter	Generate setter methods for properties

`TJSONToPascalOption` defines various options that control how Pascal code is generated from JSON data:

jpoUseSetter Generate setter methods for properties instead of direct field access

jpoGenerateLoad Generate LoadFromJSON methods for deserializing JSON data

jpoUnknownLoadPropsError Raise an error when unknown properties are encountered during loading

jpoDelphiJSON Generate code compatible with Delphi's JSON framework

jpoLoadCaseInsensitive Make JSON property loading case-insensitive

jpoGenerateSave Generate SaveToJSON methods for serializing objects to JSON

```
TJSONToPascalOptions = Set of TJSONToPascalOption
```

`TJSONToPascalOptions` is a set type based on TJSONToPascalOption (889). It allows combining multiple options to control the code generation process.

27.5 EJSONToPascal

27.5.1 Description

`EJSONToPascal` is raised when errors occur during JSON to Pascal code generation. It inherits from `EJSON` to maintain compatibility with the JSON handling framework.

27.6 TJSONToPascal

27.6.1 Description

`TJSONToPascal` is the core component that generates Object Pascal class definitions from JSON data structures. It analyzes the JSON input and creates complete Pascal unit code with classes that can serialize to and from JSON format.

The class provides comprehensive control over the code generation process through various properties and options. Generated classes include appropriate property declarations, constructor/destructor methods, and optional JSON serialization methods.

27.6.2 Method overview

Page	Method	Description
890	Create	Creates a new TJSONToPascal instance
890	Destroy	Destroys the TJSONToPascal instance
891	Execute	Generates Pascal code from JSON data

27.6.3 Property overview

Page	Properties	Access	Description
891	Active	rw	Automatically execute code generation when set to True
892	Code	rw	Generated Pascal unit code
893	DefaultParentName	rw	Default base class for generated objects
893	DestUnitName	rw	Name of the generated Pascal unit
893	ExtraUnitNames	rw	Additional units for the uses clause
894	FieldPrefix	rw	Prefix for private field names
893	IndentSize	rw	Number of spaces for code indentation
892	JSON	rw	JSON string to generate code from
891	JSONData	rw	JSON data object to generate code from
891	JSONStream	rw	Stream containing JSON data to generate code from
894	ObjectConstructorArguments	rw	Arguments for object constructors
892	Options	rw	Code generation options
892	PropertyMap	rw	Manual type mappings for JSON properties
894	PropertyTypeSuffix	rw	Suffix for generated object and array type names
894	ToplevelObjectClassName	rw	Name of the main generated class

27.6.4 TJSONToPascal.Create

Synopsis: Creates a new TJSONToPascal instance

Declaration: `constructor Create(AOwner: TComponent);  Override`

Visibility: public

Description: Creates a new `TJSONToPascal` instance and initializes default values for properties. The `AOwner` parameter specifies the component owner for memory management.

27.6.5 TJSONToPascal.Destroy

Synopsis: Destroys the TJSONToPascal instance

Declaration: `destructor Destroy;  Override`

Visibility: public

Description: Cleans up resources and destroys the `TJSONToPascal` instance. Frees the internal property map and code strings.

27.6.6 TJSONToPascal.Execute

Synopsis: Generates Pascal code from JSON data

Declaration: `procedure Execute`

Visibility: public

Description: `Execute` is the main method that performs the JSON to Pascal code generation. It analyzes the JSON data from JSONData (891), JSONStream (891), or JSON (892) properties and generates complete Pascal unit code.

The generated code is stored in the Code (892) property and includes class declarations, property definitions, and optional serialization methods based on the configured options.

27.6.7 TJSONToPascal.JSONData

Synopsis: JSON data object to generate code from

Declaration: `Property JSONData : TJSONData`

Visibility: public

Access: Read,Write

Description: `JSONData` specifies the `TJSONData` object containing the JSON structure to convert to Pascal code. This property takes priority over JSONStream (891) and JSON (892) properties.

27.6.8 TJSONToPascal.JSONStream

Synopsis: Stream containing JSON data to generate code from

Declaration: `Property JSONStream : TStream`

Visibility: public

Access: Read,Write

Description: `JSONStream` specifies a stream containing JSON data to convert to Pascal code. JSONData (891) property takes priority over this property if both are set.

27.6.9 TJSONToPascal.Active

Synopsis: Automatically execute code generation when set to True

Declaration: `Property Active : Boolean`

Visibility: published

Access: Read,Write

Description: When `Active` is set to `True`, the Execute (891) method is automatically called. This property is useful for generating code at design time in the IDE.

27.6.10 TJSONToPascal.Options

Synopsis: Code generation options

Declaration: `Property Options : TJSONToPascalOptions`

Visibility: published

Access: Read,Write

Description: `Options` controls various aspects of the code generation process using TJSONToPascalOptions (889). Common combinations include generating load/save methods or making the code compatible with different JSON frameworks.

27.6.11 TJSONToPascal.JSON

Synopsis: JSON string to generate code from

Declaration: `Property JSON : TJSONStringType`

Visibility: published

Access: Read,Write

Description: `JSON` specifies the JSON data as a string to convert to Pascal code. JSONData (891) and JSONStream (891) properties take priority over this property.

27.6.12 TJSONToPascal.Code

Synopsis: Generated Pascal unit code

Declaration: `Property Code : TStrings`

Visibility: published

Access: Read,Write

Description: `Code` contains the generated Pascal unit code after calling Execute (891). The code includes complete class definitions with properties, methods, and optional JSON serialization support.

27.6.13 TJSONToPascal.PropertyMap

Synopsis: Manual type mappings for JSON properties

Declaration: `Property PropertyMap : TPropertyMap`

Visibility: published

Access: Read,Write

Description: `PropertyMap` allows manual control over how specific JSON properties are converted to Pascal types. After Execute (891), it contains all detected and generated type mappings for properties.

27.6.14 TJSONToPascal.DestUnitName

Synopsis: Name of the generated Pascal unit

Declaration: `Property DestUnitName : string`

Visibility: published

Access: Read,Write

Description: `DestUnitName` specifies the unit name that appears in the generated Pascal code. This becomes the unit declaration in the generated code.

27.6.15 TJSONToPascal.DefaultParentName

Synopsis: Default base class for generated objects

Declaration: `Property DefaultParentName : string`

Visibility: published

Access: Read,Write

Description: `DefaultParentName` specifies the base class used for generated Pascal classes. Default is 'TObject'. Can be overridden per property using the property map.

27.6.16 TJSONToPascal.IndentSize

Synopsis: Number of spaces for code indentation

Declaration: `Property IndentSize : Integer`

Visibility: published

Access: Read,Write

Description: `IndentSize` controls how many spaces are used for indenting the generated Pascal code. Default value is 2 spaces per indentation level.

27.6.17 TJSONToPascal.ExtraUnitNames

Synopsis: Additional units for the uses clause

Declaration: `Property ExtraUnitNames : string`

Visibility: published

Access: Read,Write

Description: `ExtraUnitNames` specifies additional unit names (comma-separated) to include in the interface uses clause of the generated unit. Useful for including custom base classes or additional functionality.

27.6.18 TJSONToPascal.PropertyTypeSuffix

Synopsis: Suffix for generated object and array type names

Declaration: `Property PropertyTypeSuffix : string`

Visibility: published

Access: Read,Write

Description: `PropertyTypeSuffix` is appended to object and array type names during property map construction. Helps avoid naming conflicts and provides consistent naming conventions.

27.6.19 TJSONToPascal.FieldPrefix

Synopsis: Prefix for private field names

Declaration: `Property FieldPrefix : string`

Visibility: published

Access: Read,Write

Description: `FieldPrefix` specifies the prefix used for private field names in generated classes. Common values are 'F' (e.g., 'FName') or '_' (e.g., '_name').

27.6.20 TJSONToPascal.ObjectConstructorArguments

Synopsis: Arguments for object constructors

Declaration: `Property ObjectConstructorArguments : string`

Visibility: published

Access: Read,Write

Description: `ObjectConstructorArguments` specifies the constructor arguments inserted literally between parentheses in generated constructor calls. Useful when the parent class constructor requires specific parameters.

27.6.21 TJSONToPascal.ToplevelObjectClassName

Synopsis: Name of the main generated class

Declaration: `Property ToplevelObjectClassName : string`

Visibility: published

Access: Read,Write

Description: `ToplevelObjectClassName` specifies the class name for the main object generated from the root JSON object. This is typically the primary class users will instantiate.

27.7 TPropertyMap

27.7.1 Description

`TPropertyMap` is a collection that holds TPropertyMapItem (896) instances. It provides methods to manage mappings between JSON paths and Pascal type information during code generation.

The collection allows you to customize how specific JSON properties are converted to Pascal code, override type names, and control the generation process for individual properties.

27.7.2 Method overview

Page	Method	Description
895	AddPath	Adds a new property mapping for the specified path
895	FindPath	Finds the property mapping for the specified path
895	IndexOfPath	Returns the index of the mapping for the specified path

27.7.3 Property overview

Page	Properties	Access	Description
896	Map	rw	Indexed access to property mapping items

27.7.4 TPropertyMap.AddPath

Synopsis: Adds a new property mapping for the specified path

Declaration:
```
function AddPath(const APath: string; const ATypeName: string)
                    : TPropertyMapItem
```

Visibility: public

Description: `AddPath` creates a new TPropertyMapItem (896) for the given JSON path and type name. Returns the created mapping item for further configuration.

27.7.5 TPropertyMap.IndexOfPath

Synopsis: Returns the index of the mapping for the specified path

Declaration:
```
function IndexOfPath(const APath: string) : Integer
```

Visibility: public

Description: `IndexOfPath` searches for a property mapping with the given JSON path and returns its index. Returns -1 if no mapping is found for the specified path.

27.7.6 TPropertyMap.FindPath

Synopsis: Finds the property mapping for the specified path

Declaration:
```
function FindPath(const APath: string) : TPropertyMapItem
```

Visibility: public

Description: `FindPath` searches for and returns the TPropertyMapItem (896) for the given JSON path. Returns `nil` if no mapping is found for the specified path.

27.7.7 TPropertyMap.Map

Synopsis: Indexed access to property mapping items

Declaration: `Property Map[Aindex: Integer]: TPropertyMapItem; default`

Visibility: public

Access: Read,Write

Description: `Map` provides indexed access to the TPropertyMapItem (896) instances in the collection. This is the default property, allowing array-style access to mappings.

27.8 TPropertyMapItem

27.8.1 Description

`TPropertyMapItem` represents a mapping between a JSON path and Pascal type information. It stores details about how a specific JSON property should be converted to Pascal code, including type names, property names, and generation options.

Each item corresponds to a JSON property path and contains information about the Pascal type to generate, the parent class, and various options that control code generation for that specific property.

27.8.2 Method overview

Page	Method	Description
896	Assign	Assigns values from another TPropertyMapItem

27.8.3 Property overview

Page	Properties	Access	Description
897	Generated	r	Indicates if this mapping was automatically generated
898	JSONType	rw	JSON type for this property
897	ParentTypeName	rw	Base class name for generated objects
897	Path	rw	JSON path for this property mapping
898	PropertyName	rw	Pascal property name
898	SkipType	rw	Skip generating class or array type for this property
897	TypeName	rw	Pascal type name for this property

27.8.4 TPropertyMapItem.Assign

Synopsis: Assigns values from another TPropertyMapItem

Declaration: `procedure Assign(Source: TPersistent);  Override`

Visibility: public

Description: `Assign` copies all property values from the source `TPropertyMapItem` to this instance. Overrides the standard `TPersistent.Assign` method.

27.8.5 TPropertyMapItem.Generated

Synopsis: Indicates if this mapping was automatically generated

Declaration: `Property Generated : Boolean`

Visibility: public

Access: Read

Description: `Generated` is `True` if this property mapping was automatically created during code generation, `False` if it was manually added. Read-only property.

27.8.6 TPropertyMapItem.Path

Synopsis: JSON path for this property mapping

Declaration: `Property Path : string`

Visibility: published

Access: Read,Write

Description: `Path` specifies the JSON path that this mapping applies to. Paths use dot notation, e.g., 'person.address.street' for nested properties.

27.8.7 TPropertyMapItem.TypeName

Synopsis: Pascal type name for this property

Declaration: `Property TypeName : string`

Visibility: published

Access: Read,Write

Description: `TypeName` specifies the Pascal type name to use for this property. Can be a simple type like 'String' or 'Integer', or a class name for objects.

27.8.8 TPropertyMapItem.ParentTypeName

Synopsis: Base class name for generated objects

Declaration: `Property ParentTypeName : string`

Visibility: published

Access: Read,Write

Description: `ParentTypeName` specifies the base class to use when this property generates an object type. Overrides the DefaultParentName (893) setting for this specific property.

27.8.9 TPropertyMapItem.PropertyName

Synopsis: Pascal property name

Declaration: `Property PropertyName : string`

Visibility: published

Access: Read,Write

Description: `PropertyName` specifies the name to use for the Pascal property. If empty, the name is derived from the JSON property name.

27.8.10 TPropertyMapItem.JSONType

Synopsis: JSON type for this property

Declaration: `Property JSONType : TJSONtype`

Visibility: published

Access: Read,Write

Description: `JSONType` indicates the JSON type detected or assigned for this property. Uses the `TJSONType` enumeration (jtString, jtNumber, jtBoolean, jtArray, jtObject, etc.).

27.8.11 TPropertyMapItem.SkipType

Synopsis: Skip generating class or array type for this property

Declaration: `Property SkipType : Boolean`

Visibility: published

Access: Read,Write

Description: When `SkipType` is `True`, no class or array type declaration is generated for this property. Useful when you want to handle complex types manually or use existing types.

Chapter 28

Reference for unit 'fpmimetypes'

28.1 Used units

Table 28.1: Used units by unit 'fpmimetypes'

Name	Page
Classes	??
Contnrs	284
System	??
sysutils	??

28.2 Overview

The fpmimetypes unit contains a class which handles mapping of filename extensions to MIME (Multipurpose Internet Mail Extensions) types. The TFPMimeTypes (900) class handles this mapping. A global instance of this class is available through the MimeTypes (899) function. The list of known mime types can be instantiated through the LoadKnownTypes (901) method, or a file in the standard mime.types format can be loaded through the LoadFromFile (902) method.

28.3 Procedures and functions

28.3.1 MimeTypes

Synopsis: Global `TFPMimeTypes` instance.

Declaration: `function MimeTypes : TFPMimeTypes`

Visibility: default

Description: `MimeTypes` returns a global instance of the TFPMimeTypes (900) class. It is not initialized with a list of extensions, so this instance must still be explicitly initialized with TFPMimeTypes.LoadKnownTypes (901)

This function is not thread-safe, so be sure to call it once from the main thread and initialize the resulting list.

See also: TFPMimeTypes (900), TFPMimeTypes.LoadKnownTypes (901)

28.4 TFPMimeTypes

28.4.1 Description

`TFPMimeTypes` manages a list of MIME types.

The list of types can be initialized with the OS list of known MIME types through the LoadKnownTypes (901) method, or a file in the standard mime.types format can be loaded through the LoadFromFile (902) method.

The associated mime type of a file extension can be retrieved with TFPMimeTypes.GetMimeType (902).

See also: TFPMimeTypes.LoadKnownTypes (901), TFPMimeTypes.LoadFromFile (902), TFPMimeTypes.GetMimeType (902)

28.4.2 Method overview

Page	Method	Description
902	AddType	Add a MIME type to the list.
901	Clear	Clear the list of known `MIME` types.
900	Create	Create a new instance of the `TFPMimeTypes` class.
900	Destroy	Remove instance from memory.
903	GetKnownExtensions	Get a lis of all known extensions.
903	GetKnownMimeTypes	Get a list of all known MIME types.
902	GetMimeExtensions	Get the extensions associated with a MIME type.
902	GetMimeType	Get MIME type of an extension.
901	GetNextExtension	Extract an extension from an extension list as returned by GetMimeExtensions.
902	LoadFromFile	Load mime types from a file in mime.types file format.
901	LoadFromStream	Load mime types from a stream in mime.types file format.
901	LoadKnownTypes	Queries the OS for a list of known MIME types.

28.4.3 TFPMimeTypes.Create

Synopsis: Create a new instance of the `TFPMimeTypes` class.

Declaration:
```
constructor Create(AOwner: TComponent);  Override
```

Visibility: public

Description: The `Create` method sets up the necessary internal structures.

See also: TFPMimeTypes.Destroy (900)

28.4.4 TFPMimeTypes.Destroy

Synopsis: Remove instance from memory.

Declaration:
```
destructor Destroy;  Override
```

Visibility: public

Description: `Destroy` destroys the `TFPMimeTypes` instance and removes it from memory.

See also: TFPMimeTypes.Create (900)

28.4.5 TFPMimeTypes.Clear

Synopsis: Clear the list of known MIME types.

Declaration: `procedure Clear`

Visibility: public

Description: `Clear` clears the list of known mime types.

See also: TFPMimeTypes.LoadKnownTypes (901)

28.4.6 TFPMimeTypes.LoadKnownTypes

Synopsis: Queries the OS for a list of known MIME types.

Declaration: `procedure LoadKnownTypes;  Virtual`

Visibility: public

Description: `LoadKnownTypes` uses the default mechanism of the OS to initialize the list of MIME types. Under windows, this loads a list of known extensions from the registry (under `HKEY_CLASSES_ROOT`) and attempts to load a mime.types located next to the application binary. Under unixlike OS-es, the system location for the mime.types is used to load the mime.types file.

See also: TFPMimeTypes.LoadFromFile (902)

28.4.7 TFPMimeTypes.GetNextExtension

Synopsis: Extract an extension from an extension list as returned by GetMimeExtensions.

Declaration: `class function GetNextExtension(var E: string) : string`

Visibility: public

Description: `GetNextExtension` is a helper function that extracts adn returns the next extension from the semicolon separated list of extensions `E` and removes the extension from the list. If there is no more extension, the empty string is return.

See also: TFPMimeTypes.GetMimeExtensions (902)

28.4.8 TFPMimeTypes.LoadFromStream

Synopsis: Load mime types from a stream in mime.types file format.

Declaration: `procedure LoadFromStream(const Stream: TStream);  Virtual`

Visibility: public

Description: `LoadFromStream` parses the stream for MIME type definitions and extensions and adds them to the list of known MIME types. The stream is expected to have the text format of the mime.types as found in unix systems.

See also: TFPMimeTypes.LoadFromFile (902)

28.4.9 TFPMimeTypes.LoadFromFile

Synopsis: Load mime types from a file in mime.types file format.

Declaration: `procedure LoadFromFile(const AFileName: string)`

Visibility: public

Description: `LoadFromFile` loads the file `aFileName` and parses the file for MIME type definitions and extensions and adds them to the list of known MIME types. The file is expected to have the text format of the mime.types as found in unix systems.

Errors: if the file `aFileName` does not exist, an exception is raised.

See also: TFPMimeTypes.LoadFromStream (901)

28.4.10 TFPMimeTypes.AddType

Synopsis: Add a MIME type to the list.

Declaration: `procedure AddType(const AMimeType: string; const AExtensions: string)`

Visibility: public

Description: `AddType` can be used to add `AMimeType` to the list of known MIME types, and associate a semicolon-separated list of extensions `AExtensions` with it. If `AMimeType` is already present in the list of MIME types, then the list of extensions in `AExtensions` is merged with the existing extensions. The extensions may not have a dot character prepended to them.

See also: TFPMimeTypes.GetMimeExtensions (902), TFPMimeTypes.GetMimeType (902)

28.4.11 TFPMimeTypes.GetMimeExtensions

Synopsis: Get the extensions associated with a MIME type.

Declaration: `function GetMimeExtensions(const AMimeType: string) : string`

Visibility: public

Description: `GetMimeExtensions` returns the list of extensions associated with a MIME type (`AMimeType`). If none are known, an empty string is returned. `AMimeType` is searched case insentively.

See also: TFPMimeTypes.GetMimeType (902)

28.4.12 TFPMimeTypes.GetMimeType

Synopsis: Get MIME type of an extension.

Declaration: `function GetMimeType(const AExtension: string) : string`

Visibility: public

Description: `GetMimeType` returns the MIME type of the extension `AExtension`. The extension is searched case-insensitive. If no MIME type is found, an empty string is returned. The extension may start with a dot character or not.

See also: TFPMimeTypes.GetMimeExtensions (902)

28.4.13 TFPMimeTypes.GetKnownMimeTypes

Synopsis: Get a list of all known MIME types.

Declaration: `function GetKnownMimeTypes(AList: TStrings) : Integer`

Visibility: public

Description: `GetKnownMimeTypes` fills `AList` with the list of known MIME types (one per line) in random order. It clears the list first, and returns the number of entries added to the list.

See also: TFPMimeTypes.GetKnownExtensions (903)

28.4.14 TFPMimeTypes.GetKnownExtensions

Synopsis: Get a lis of all known extensions.

Declaration: `function GetKnownExtensions(AList: TStrings) : Integer`

Visibility: public

Description: `GetKnownExtensions` fills `AList` with the list of known extensions (one per line) in random order. It clears the list first, and returns the number of entries added to the list. The extensions do not have a dot (.) character prepended.

See also: TFPMimeTypes.GetKnownMimeTypes (903)

28.5 TMimeType

28.5.1 Description

`TMimeType` is a helper class which stores a MIME type and it's associated extensions. It should not be necessary to create instances of this class manually, the creation is handled entirely through the TFPMimeTypes (900) class.

See also: TMimeType.MimeType (904), TMimeType.Extensions (904), TFPMimeTypes (900)

28.5.2 Method overview

Page	Method	Description
904	Create	Create a new instance of `TMimeType`.
904	MergeExtensions	Merge extensions in the list of extensions.

28.5.3 Property overview

Page	Properties	Access	Description
904	Extensions	rw	Semicolon-separated list of extensions associated with `MimeType`.
904	MimeType	rw	Mime type name.

28.5.4 TMimeType.Create

Synopsis: Create a new instance of `TMimeType`.

Declaration: `constructor Create(const AMimeType: string; const AExtensions: string)`

Visibility: public

Description: `Create` initializes a new instance of `TMimeType` and sets the TMimeType.MimeType (904)TMimeType.Extensions (904) properties to `aMimeType` and `aExtensions`.

See also: TMimeType.MimeType (904), TMimeType.Extensions (904)

28.5.5 TMimeType.MergeExtensions

Synopsis: Merge extensions in the list of extensions.

Declaration: `procedure MergeExtensions(AExtensions: string)`

Visibility: public

Description: `MergeExtensions` merges the comma-separated list of extensions in `AExtensions` into TMimeType.Extensions (904) in such a way that there are no duplicates.

See also: TMimeType.Extensions (904)

28.5.6 TMimeType.MimeType

Synopsis: Mime type name.

Declaration: `Property MimeType : string`

Visibility: public

Access: Read,Write

Description: `MimeType` is the lowercase name of the mime type.

See also: TMimeType.Extensions (904)

28.5.7 TMimeType.Extensions

Synopsis: Semicolon-separated list of extensions associated with `MimeType`.

Declaration: `Property Extensions : string`

Visibility: public

Access: Read,Write

Description: `Extensions` is the comma-separated list of extensions that is associated with MimeType (904)

See also: TMimeType.MimeType (904)

Chapter 29

Reference for unit 'fpmkunit'

29.1 Used units

Table 29.1: Used units by unit 'fpmkunit'

Name	Page
BaseUnix	??
Classes	??
cthreads	??
libtar	1351
process	1420
System	??
sysutils	??
Zipper	1651
ZStream	1690

29.2 Overview

The `fpmkunit` unit implements the core building and installation engine for Free Pascal packages. It provides a complete set of classes and functions for managing package compilation, dependency resolution, file installation, and archive creation.

This unit forms the foundation of the fpmake build system, offering classes like TPackage (1009) for package management, TTarget (1049) for build targets, TDependency (988) for dependency tracking, and various installer classes for deployment.

The building engine supports cross-platform compilation with CPU and OS-specific conditional compilation, package variants, custom commands, and flexible installation schemes.

29.3 Usage description

The `fpmkunit` unit is meant to be used to create a program that handles compilation, archiving and installation of a set of pascal sources, collected in so-called packages. The unit can handle multiple packages, and will manage dependencies of packages automatically. Customarily this program is called fpmake. When using fppkg or in the FPC sources, use of this name is mandatory.

The unit supports the following operations:

- Compilation of units, programs or libraries.
- installation of the compiled results to standard locations.
- Creation of a .zip file with the sources of the files.
- Generation of documentation with fpdoc.
- Execution of custom commands when compiling, installing or packaging

The unit consists of a series of classes which are designed to work together. An instance of these classes is created for you, they are ready to be used.

Installer This function returns the main TCustomInstaller (983) instance. This class is the entry point for all functionality: it manages settings and package definitions and processes command-line options

Defaults The main TCustomDefaults (967) Instance. Here you can set compiler options such as CPU, OS, all kinds of search paths, executable paths and directories

Installer.BuildEngine This is the engine that does all the work.

For the majority of cases, it suffices to define some packages using Installer.Packages (985), and to call Installer.Run (984). The `Run` method will examine the RunMode (985) property to determine what action is expected, and then it proceeds to execute the action using the BuildEngine (985). The latter does all the work, and contains all the logic to launch commands in the correct order to execute the specified action.

To demonstrate this, a simple program can be made:

```
program fpmake1;

uses fpmkunit;

var
  P: TPackage;

begin
  With Installer do
    begin
    P:=Packages.AddPackage('mypackage');
    P.Dependencies.Clear;
    P.Targets.AddUnit('src/unit1.pp');
    Run;
    end;
end.
```

The TCustomInstaller (983) class maintains a collection of Packages (985). At least 1 package must be defined. Every package has a list of files that must be compiled, and possibly a list of files that needs to be installed together with installed programs.

Every file that needs to be compiled, is a TTarget (1049), and all targets are in the Targets (1027) property of the TPackage (1009) class. Installer will compile the targets in the order that they are specified, unless dependencies have been set up (see also Dependencies (910)), in which case the order will be changed to compile dependencies of a target first, for any given target.

After compiling the fpmake1.pp program, it can be used to compile the unit1.pp when run with the `compile` command;

```
>  ./fpmake1 compile
Start compiling package mypackage for target x86_64-linux.
       Compiling src/unit1.pp
[100%] Compiled package mypackage
```

To pass extra options to the compiler, we can use the Defaults (932) variable, which is valid as soon as the installer is created:

```
program fpmake2;

uses fpmkunit;

var
  P: TPackage;

begin
  With Installer do
    begin
    Defaults.Options.Add('-O2');
    P:=Packages.AddPackage('mypackage');
    P.Dependencies.Clear;
    P.Targets.AddUnit('src/unit1.pp');
    Run;
    end;
end;
```

This program can be compiled, and when run with the `-v` option, you can see that the compiler is called with the extra option:

```
./fpmake2 compile -v
Start compiling package mypackage for target x86_64-linux.
  Compiling target unit1
    Executing command "/home/michael/bin/fpc  \
                       -Tlinux -Px86_64 \
                       -FUunits/x86_64-linux/ \
                       -O2 src/unit1.pp"
Generating "mypackage-x86_64-linux.fpm"
[100%] Compiled package mypackage
```

(the lines have been split for readability. On the console the log would be on one line)

To make life easier, if you are using the 3.3.1 or higher version of fpmkunit, you can use the AddPackageFromDir (983) call.

```
uses fpmkunit;

begin
  With Installer do
    begin
    AddPackageFromDir('mypackage','src');
    Run;
```

```
    end;
end.
```

When compiled and run, the output is the following:

```
> ./fpmake3 compile -v
Adding unit src/unit1.pp to targets of mypackage
Switching to buildunit compilation of package mypackage
Start compiling package mypackage for target x86_64-linux.
  Compiling target unit1
Compiling target BuildUnit_mypackage
Not using buildunit
  Executing command "/home/michael/bin/fpc \
                     -Tlinux -Px86_64 \
                     -FUunits/x86_64-linux/ \
                     -Fusrc \
                     units/x86_64-linux/BuildUnit_mypackage.pp"
Deleted file "units/x86_64-linux/BuildUnit_mypackage.ppu"
Deleted file "units/x86_64-linux/BuildUnit_mypackage.pp"
Generating "mypackage-x86_64-linux.fpm"
[100%] Compiled package mypackage
```

Note that the installer switched to using a build unit: the AddPackageFromDir (905) command does not check dependencies of the units. Instead, it switches to using a build unit (a unit that simply uses all units to be compiled) and then compiles the buildunit: in that case, the compiler will compile all files in the correct order.

See also: CommandLineOptions (908), Dependencies (910)

29.4 Commandline options

The `fpmkunit` unit takes control of parsing the command-line to decide what to do and what options to set. The command-line takes a mandatory command name followed by options controlling the behaviour.

```
fpmake command [options]
```

The `command` must be one of the following:

compile Compile all units in the selected package or packages.

build Force a build of all units in the selected package or packages.

install Install all units in the selected package or packages.

buildinstall Build and install all units in the selected package or packages.

uninstall Uninstall the selected package or packages.

clean Remove all generated files in the selected package or packages for current CPU-OS target.

distclean Remove all generated files in the selected package or packages for all targets.

archive Create archive (zip) with all units in the selected package or packages.

manifest Create a manifest suitable for import in repository.

zipinstall Install all units in the selected package or packages into an archive.

pkglist Create list of all packages suitable for FPC installer.

fpdocproject Create fpdoc project file(s) for all packages

The behaviour of the command can be controlled by specifying one of the many options:

-h or -help Shows a help message

-lc-list-commands list commands instead of actually executing them.

-n-nofpccfg Compiler will not use fpc.cfg

-v-verbose Be verbose when working.

-d-debug Add debug information when working.

-I-interactive Allow to interact with child processes.

-e-useenv Use environment to pass options to compiler.

-ie-installexamples Install the example-sources.

-bu-buildunit Compile package in Build-unit mode.

-sap-skipallprograms Skip all programs even if native-compiling/installing.

-scp-skipcrossprograms Skip programs when cross-compiling/installing.

-io-ignoreinvalidoption Ignore further invalid options.

-C-cpu=Value Compile for indicated CPU.

-O-os=Value Compile for indicated OS.

-s-subtarget=Value Compile for indicated subtarget.

-t-target=Value Compile for indicated target, string in the form CPU-OS(-subtarget).

-P-prefix=Value Use indicated prefix directory for all commands.

-B-baseinstalldir=Value Use indicated directory as base install dir.

-BI-bininstalldir=Value Use indicated directory as base install dir.

-LI-libinstalldir=Value Use indicated directory as base install dir.

-UL-localunitdir=Value Use indicated directory as local (user) unit dir.

-UG-globalunitdir=Value Use indicated directory as global unit dir.

-sp-searchpath=Value Add search directory for packages.

-U-unitinstalldir=Value Use indicated directory to install units into.

-r-compiler=Value Use indicated binary as compiler.

-f-config=Value Use indicated config file when compiling.

-o-options=Value Pass extra options to the compiler.

-df-doc-folder=Value Use indicated directory as fpdoc output folder.

-sd-single-docfile=Value Create a single fpdoc project file for all projects.

-do-doc-options=Value Name=Value File with options for fpdoc project file.

-fsp-fpunitsrcpath=Value Sourcepath to replace in fpunits.cfg on installation.

-zp-zipprefix=Value Use indicated prefix for generated archives.

-T-threads=Value Enable the indicated amount of worker threads.

-data2inc=Value Use indicated data2inc executable.

Packages can come in variants. To add a package-variant to all packages:

```
+[variantname]+=[variant1],variant2,...
```

To add a package-variant to all packages which other packages will inherit:

```
+[variantname]*=[variant1],variant2,...
```

To add specific options for one package-variant:

```
--options_[variantname]_[variant1]=Value
```

See also: Usage (905), Dependencies (910)

29.5 Handling dependencies

When compiling units and programs, the installer will by default compile all targets in the order that they were specified

However, if `unita` depends on `unitb`, and `unita` was specified before `unitb`, then `unitb` will be compiled twice: when compiling `unita` the compiler will compile it because it needs `unitb`. Then `unitb2` will be compiled because it is in the list of targets. Clearly, this is not ideal or efficient.

There are 2 ways to solve this:

1. Specify the dependencies of the units. In this case, when a target has dependencies, the installer will compile dependencies first, and will compile the target last
2. Switch to using a "build unit". This unit is an empty unit which simply lists all units to be compiled. Instead of compiling the units one by one, the installer will try to compile the buildunit, and then the compiler will make sure that the units are compiled in the correct order, so every unit is compiled only once (for interdependent units the compiler may still compile them twice).

The latter is clearly the easier solution. When using a build unit, the installer will of course delete the unit sources and `.ppu` file after compiling, since they are not needed any more after compilation.

When running the installer, you can request that it tries to use a build-unit by giving the `-bu` or `-buildunit` command-line option. However, there are some cases when using a build unit is not an option. Therefore, the TPackage.SupportedBuildModes (1009) property can be used to indicate which build modes can be used for a package. if the `bmBuildUnit` build mode (TBuildMode (917)) is not in the set of buildmodes, then the installer will compile one by one, even when a buildunit was requested.

To specify dependencies, you can use the Dependencies (1053) property of TTarget (1049):

```
uses fpmkunit;

var
  P: TPackage;
  T : TTarget;

begin
  With Installer do
    begin
    P:=Packages.AddPackage('mypackage');
    P.Dependencies.Clear;
    T:=P.Targets.AddUnit('src2/unita.pp');
    T.Dependencies.AddUnit('unitb');
    P.Targets.AddUnit('src2/unitb.pp');
    Run;
    end;
end.
```

You can add a unit to the dependencies using TDependencies.AddUnit (987), or a complete package using TDependencies.Add (987).

When compiling the above and running it, the following output can be seen

```
./fpmake4 compile -v
Start compiling package mypackage for target x86_64-linux.
      Compiling target unitb
        Executing command "/home/michael/bin/fpc \
                           -Tlinux -Px86_64 \
                           -FUunits/x86_64-linux/ \
                           src2/unitb.pp"
  Compiling target unita
    Executing command "/home/michael/bin/fpc \
                       -Tlinux -Px86_64 \
                       -FUunits/x86_64-linux/ \
                       src2/unita.pp"
Generating "mypackage-x86_64-linux.fpm"
[100%] Compiled package mypackage
```

As you can see, the installed compiled `unitb` first.

See also: CommandLineOptions (908), Usage (905)

29.6 Constants, types and variables

29.6.1 Constants

```
AIXSharedLibExt = '.a'
```

Constant defining the file extension for shared libraries on IBM AIX systems. AIX uses specific conventions for shared library naming and loading.

```
AllAmigaLikeOSes = [Amiga, MorphOS, AROS]
```

Constant set containing operating systems that are compatible with or derived from AmigaOS. Includes classic AmigaOS and modern Amiga-compatible systems.

```
AllBSDOSes = [FreeBSD, NetBSD, OpenBSD, Darwin, iphonesim, ios, dragonfly
  ]
```

Constant set containing all Berkeley Software Distribution (BSD) operating system variants. Includes FreeBSD, OpenBSD, NetBSD, and Darwin/macOS.

```
AllCPUs = [Low(TCPU)..High(TCPU)]
```

Constant set containing all CPU architectures supported by the Free Pascal Compiler. Used for platform-independent operations or when checking CPU compatibility across all architectures.

```
AllImportLibraryOSes = AllWindowsOSes + [os2, emx, netwlibc, netware
  , watcom, go32v2, macosclassic, nativent, msdos, win16]
```

Constant set containing operating systems that require import libraries for dynamic linking. Import libraries contain stubs for symbols exported from dynamic libraries.

```
AllLimit83fsOses = [go32v2, os2, emx, watcom, msdos, win16, atari
  ]
```

Constant set containing operating systems that enforce the traditional 8.3 filename format limitation. Used to determine when filenames need to be truncated or adjusted for compatibility.

```
AllMessages = [vlError, vlWarning, vlCommand, vlInfo]
```

Constant set containing all available message verbosity levels. Includes debug, informational, warning, and error messages for complete logging.

```
AllOSes = [Low(TOS)..High(TOS)]
```

A predefined set that includes all operating systems supported by the Free Pascal Compiler. This constant is used for targets that should be compiled for all available platforms.

```
AllSmartLinkLibraryOSes = [Linux, msdos, win16, palmos]
```

Constant set containing operating systems that support smart linking of libraries. Smart linking allows the linker to include only used symbols, reducing executable size.

```
AllUnixOSes = [Linux, FreeBSD, NetBSD, OpenBSD, Darwin, QNX, BeOS
  , Solaris, Haiku, iphonesim, ios, aix, Android, dragonfly]
```

Constant set containing all operating systems that follow Unix conventions and standards. Includes Linux, BSD variants, and other POSIX-compatible systems.

```
AllWindowsOSes = [Win32, Win64, WinCE]
```

Constant set containing all Microsoft Windows operating system versions supported by Free Pascal. Includes desktop and server editions across different Windows generations.

```
Amd64 = X86_64
```

Constant representing the AMD64/x86_64 CPU architecture in the FPMake build system. This 64-bit architecture is commonly used on modern desktop and server systems from both AMD and Intel, providing extended memory addressing capabilities and improved performance over 32-bit architectures.

```
DbgExt = '.dbg'
```

Constant defining the file extension for debug information files. These files contain debugging symbols and information for debugger support.

```
DefaultMessages = [vlError, vlWarning, vlCommand]
```

Constant set defining the default verbosity levels for build system messages. Includes error and warning messages that are typically displayed to users.

```
DirNotFound = '<dirnotfound>'
```

Constant representing the error code returned when a directory cannot be found. Used for consistent error handling in directory operations throughout the build system.

```
DLLExt = '.dll'
```

Constant defining the file extension for Dynamic Link Libraries on Windows ('.dll'). DLLs contain code and resources that can be used by multiple programs simultaneously.

```
DocProjectFileExt = '-docs.xml'
```

Constant defining the file extension for FPDoc documentation project files. These files configure how documentation is generated from source code comments.

```
DOS = Go32v2
```

Represents the DOS operating system in the FPMake build system.

```
DyLibExt = '.dylib'
```

Constant defining the file extension for dynamic libraries on macOS ('.dylib'). These libraries provide shared functionality on Apple platforms and are loaded at runtime.

```
ExeExt = '.exe'
```

Constant defining the platform-specific file extension for executable files. On Windows this is '.exe', while Unix-like systems typically use no extension.

```
FPMakePPFile = 'fpmake.pp'
```

Constant defining the standard filename for FPMake Pascal program files ('fpmake.pp'). This is the default name for the main build script file in Free Pascal packages.

```
FpmkExt = '.fpm'
```

Constant defining the file extension for FPMake build script files ('.fpmk'). These files contain build configuration and packaging instructions for Free Pascal projects.

```
IncExt = '.inc'
```

Constant defining the file extension for Pascal include files ('.inc'). These files contain source code that is included into other Pascal files using include directives.

```
LibExt = '.a'
```

Constant defining the file extension for static library files. Static libraries contain precompiled object code that is linked directly into executables at compile time.

```
MacOSX = Darwin
```

Represents the macOS (formerly Mac OS X) operating system in the FPMake build system.

```
ManifestFile = 'manifest.xml'
```

Constant defining the filename for package manifest files. Manifest files contain metadata about package contents, dependencies, and installation requirements.

```
ObjExt = '.o'
```

Constant defining the file extension for compiled object files. Object files contain machine code generated from source compilation and are linked together to create executables.

```
OSCPUSupported : Array[TOS,TCpu] of Boolean = ((False, False, False
  , False, False, False, False, False, False, False, False, False, False
  , False, False, False), (False, True, True, True, True, True, True
  , True, False, True, True, True, False, False, True, True), (False
  , True, False, False, False, False, False, False, False, False, False
  , False, False, False, False, False), (False, True, False, False,
  False, False, False, False, False, False, False, False, False, False
  , False, False), (False, True, False, False, False, False, False,
  False, False, False, False, False, False, False, False, False),
  (False, True, True, False, False, True, False, False, False, False
  , False, False, False, False, False, False), (False, True, False,
  False, False, False, False, False, False, False, False, False, False
  , False, False, False), (False, True, True, True, True, True, True
  , False, False, False, False, False, False, False, False, False),
  (False, False, True, True, False, False, False, False, False, False
  , False, False, False, False, False, False), (False, False, True,
  False, False, False, False, False, False, False, False, False, False
  , False, False, False), (False, True, False, False, True, True, False
  , False, False, False, False, False, False, False, False, False),
  (False, True, False, False, False, False, False, False, False, False
  , False, False, False, False, False, False), (False, True, False,
  False, False, False, False, False, False, False, False, False, False
  , False, False, False), (False, True, True, False, False, True, False
```

```
  , False, False, False, False, False, False, False, False, False),
  (False, True, False, False, False, False, False, False, False, False
  , False, False, False, False, False, False), (False, False, True,
  False, False, False, False, True, False, False, False, False, False, False
  , False, False, False), (False, False, True, True, False, False, False
  , False, False, False, False, False, False, False, False, False),
  (False, True, False, True, False, True, False, True, False, False
  , False, False, False, False, True, False), (False, True, False, False
  , False, False, False, False, False, False, False, False, False, False
  , False, False), (False, True, False, False, False, False, False,
  False, False, False, False, False, False, False, False, False, False),
  (False, False, False, True, False, False, False, False, False, False
  , False, False, False, False, False, False), (False, True, False,
  False, False, False, False, False, False, False, False, False, False, False
  , False, False, False), (False, False, False, False, False, True,
  False, False, False, False, False, False, False, False, False, False, False
  ), (False, True, False, False, False, False, True, False, False, False
  , False, False, False, False, False, False), (False, False, False
  , False, False, False, True, False, False, False, False, False, False, False
  , False, False, False), (False, False, False, False, False, False
  , True, False, False, False, False, False, False, False, False, False
  ), (False, True, True, True, True, True, True, True, True, True, False
  , True, False, True, False, False), (False, True, False, False, False
  , False, True, False, False, False, False, False, False, False, False
  , False), (False, True, False, False, False, True, False, False, False
  , False, False, False, False, False, False, False), (False, True,
  False, False, False, True, False, False, False, False, False, False
  , False, False, False, False), (False, False, False, True, False,
  False, False, False, True, False, False, False, False, False, False, False
  , False), (False, False, False, False, False, False, False, False
  , False, False, False, True, False, False, False), (False, True,
  False, False, False, False, True, False, False, False, False, False, False
  , False, False, False, False), (False, False, False, False, False
  , False, False, False, False, False, True, False, False), (False,
  False, False, True, False, False, False, False, False, False, False
  , False, False, False, False, False), (False, True, False, False,
  False, True, True, False, False, False, False, False, False, False
  , False, False), (False, False, False, False, False, True, False,
  False, False, False, False, False, False, False, False, False, False),
  (False, False, False, False, False, False, False, False, False, False
  , False, False, False, True, False, False), (False, False, False,
  False, False, False, True, False, False, False, False, False, False
  , False, True, False))
```

Two-dimensional array that defines which CPU architectures are supported for each operating system. Used to validate target platform combinations before attempting compilation.

```
PasExt = '.pas'
```

Constant defining the standard file extension for Pascal source files ('.pas'). Used to identify Pascal source files in package building and compilation operations.

```
PkgListFileBase = 'pkg-'
```

Constant defining the base name for package list files used by the package management system. Combined with extensions to create complete package list filenames.

```
PkgListFileExt = '.lst'
```

Constant defining the file extension for package list files. These files contain lists of available packages and their metadata.

```
PPC = PowerPC
```

Represents the PowerPC 32-bit architecture in the FPMake build system.

```
PPC64 = PowerPC64
```

Represents the PowerPC 64-bit architecture in the FPMake build system.

```
PPExt = '.pp'
```

Constant defining the file extension for Pascal program files ('.pp'). Designates main program files that contain an executable program rather than a unit.

```
PPUExt = UnitExt
```

Constant defining the file extension for Free Pascal compiled unit files ('.ppu'). These binary files contain the compiled interface and implementation of Pascal units, allowing for separate compilation and linking in the FPMake build process.

```
ProgramTargets = [ttProgram, ttExampleProgram, ttSharedLibrary]
```

Constant set containing all target types that represent executable programs. Used to filter and identify program targets in build operations.

```
RsjExt = '.rsj'
```

Constant defining the file extension for resourcestring files in JSON format ('.rsj'). These files store resource strings in JSON format for localization and internationalization.

```
RstExt = '.rst'
```

Constant defining the file extension for resource string files ('.rst'). These files contain compiled resource strings extracted from Pascal source files for localization and internationalization.

```
SharedLibExt = '.so'
```

Constant defining the platform-specific file extension for shared/dynamic libraries. Shared libraries are loaded at runtime and can be shared between multiple executables.

```
UnitExt = '.ppu'
```

Constant defining the standard file extension for Free Pascal unit source files ('.pas'). This extension is used by the FPMake system to identify and process unit source files during compilation and package building operations.

```
UnitTargets = [ttUnit, ttImplicitUnit, ttCleanOnlyUnit, ttExampleUnit
  ]
```

Constant set containing all target types that represent Pascal units. Used to filter and identify unit targets in build operations.

```
ZipExt = '.zip'
```

Constant defining the file extension for ZIP archive files ('.zip'). Used for creating and identifying package distribution archives in the fpmake system.

29.6.2 Types

```
TArchiveEvent = procedure(const AFileName: string; List: TStrings
  )
                            of object
```

`TArchiveEvent` defines the signature for event handlers that process archive operations. This event type enables custom archive processing and notification during package archiving.

Event handlers receive the archive filename and file list, allowing custom processing, validation, or modification of archive operations within the build system.

```
TArchiveProc = procedure(const AFileName: string; List: TStrings)
```

`TArchiveProc` defines the signature for procedural archive handlers that process file archiving operations. This procedure type enables pluggable archive implementations and custom archiving logic.

Archive procedures receive the target filename and file list, implementing the actual compression and archive creation functionality for the build system.

```
TBuildMode = (bmOneByOne,bmBuildUnit)
```

Table 29.2: Enumeration values for type TBuildMode

Value	Explanation
bmBuildUnit	Use a build unit
bmOneByOne	Build targets one by one sequentially

`TBuildMode` specifies how the build process should execute package compilation:

bmOneByOne specifies that build targets should be compiled sequentially, one after another. This mode relies on proper dependency specification.

bmBuildUnit specifies that individual Pascal units should not be compiled separately. Rather they are compiled as part of a larger unit compilation: the build unit.

```
TBuildModes = Set of TBuildMode
```

Set containing one or more TBuildMode (917) values.

Used to specify multiple build modes.

```
TCheckDependencyResult = (cdAvailable,cdNotAvailable,cdNotYetAvailable
  )
```

Table 29.3: Enumeration values for type TCheckDependencyResult

Value	Explanation
cdAvailable	Dependency is available and can be used
cdNotAvailable	Dependency is not available and cannot be found
cdNotYetAvailable	Dependency is not yet available but may become available later

`TCheckDependencyResult` indicates the status of dependency availability checks.

Used to determine whether required dependencies are available for compilation.

cdAvailable indicates that the required dependency is present in the system and can be used for compilation. This result allows the build process to proceed.

cdNotAvailable indicates that the required dependency cannot be found in the system and is not available for use. This result prevents the build process from proceeding due to missing dependencies.

cdNotYetAvailable indicates that the dependency is not currently available but will become available after other dependencies are built or installed

```
TCommandAt = (caBeforeCompile,caAfterCompile,caBeforeInstall,
  caAfterInstall,caBeforeArchive,caAfterArchive,
  caBeforeClean,caAfterClean,caBeforeDownload,
  caAfterDownload)
```

Table 29.4: Enumeration values for type TCommandAt

Value	Explanation
caAfterArchive	Execute command after archive creation
caAfterClean	Execute command after cleaning
caAfterCompile	Execute command after compilation
caAfterDownload	Execute command after downloading
caAfterInstall	Execute command after installation
caBeforeArchive	Execute command before archive creation
caBeforeClean	Execute command before cleaning
caBeforeCompile	Execute command before compilation
caBeforeDownload	Execute command before downloading
caBeforeInstall	Execute command before installation

Enumeration that specifies when a custom command should be executed during the build process. Determines the phase at which user-defined commands are run relative to compilation steps.

caBeforeCompile specifies that a custom command should be executed before the compilation phase begins.

caAfterCompile specifies that a custom command should be executed after compilation completes successfully.

caBeforeInstall specifies that a custom command should be executed before the installation phase begins.

caAfterInstall specifies that a custom command should be executed after installation completes successfully.

caBeforeArchive specifies that a custom command should be executed before archive creation begins.

caAfterArchive specifies that a custom command should be executed after archive creation completes.

caBeforeClean specifies that a custom command should be executed before the clean operation begins.

caAfterClean specifies that a custom command should be executed after the clean operation completes.

caBeforeDownload specifies that a custom command should be executed before download operations begin.

caAfterDownload specifies that a custom command should be executed after download operations complete successfully.

```
TCompilerMode = (cmFPC,cmTP,cmObjFPC,cmDelphi,cmMacPas)
```

Table 29.5: Enumeration values for type TCompilerMode

Value	Explanation
cmDelphi	Delphi compatibility mode
cmFPC	Standard Free Pascal mode
cmMacPas	Macintosh Pascal compatibility mode
cmObjFPC	Object Pascal mode with object-oriented features
cmTP	Turbo Pascal compatibility mode

`TCompilerMode` specifies the language compatibility mode used by the Free Pascal Compiler. Each mode affects syntax rules, default behavior, and language feature availability during compilation.

```
TCompilerModes = Set of TCompilerMode
```

`TCompilerModes` is a set type that can contain multiple TCompilerMode (919) values, allowing build configurations to specify which compiler language modes should be supported or used. This enables flexible language compatibility settings for different parts of a package build process.

```
TConditionalDestStringClass = Class of TConditionalDestString
```

TConditionalDestStringClass is a class reference type that points to TConditionalDestString (963) or its descendant classes.

This type enables polymorphic creation of different conditional destination string implementations at runtime.

It is used in factory methods and class registration systems for dynamic object instantiation.

```
TConditionalStringClass = Class of TConditionalString
```

TConditionalStringClass is a class reference type that points to TConditionalString (965) or its descendant classes.

This type is used in factory methods and class registration systems to create conditional string instances dynamically.

It enables polymorphic creation of different conditional string implementations at runtime.

```
TCopyFileProc = procedure(const APackage: TPackage;
  const ASourceFileName: string;
  const ADestFileName: string) of object
```

TCopyFileProc defines the signature for callback procedures that handle file copying operations during package installation. This procedural type enables custom file copying implementations.

The procedure receives source and destination file paths and handles the actual file copying process with appropriate error handling and progress reporting.

```
TCpu = (cpuNone,i386,m68k,powerpc,sparc,x86_64,arm,powerpc64,avr,armeb
  ,
       mips,mipsel,jvm,i8086,aarch64,sparc64)
```

Table 29.6: Enumeration values for type TCpu

Value	Explanation
aarch64	ARM 64-bit architecture
arm	ARM 32-bit architecture
armeb	ARM 32-bit big-endian architecture
avr	Atmel AVR microcontroller architecture
cpuNone	No specific CPU architecture
i386	Intel 32-bit x86 architecture
i8086	Intel 16-bit x86 architecture
jvm	Java Virtual Machine bytecode target
m68k	Motorola 68000 architecture
mips	MIPS 32-bit big-endian architecture
mipsel	MIPS 32-bit little-endian architecture
powerpc	PowerPC 32-bit architecture
powerpc64	PowerPC 64-bit architecture
sparc	SPARC 32-bit architecture
sparc64	SPARC 64-bit architecture
x86_64	Intel/AMD 64-bit x86 architecture

TCpu enumerates the CPU architectures supported by the Free Pascal Compiler for cross-platform compilation. This enumeration is used throughout the build system to specify target CPU architectures and handle CPU-specific compilation requirements.

```
TCPUS = Set of TCpu
```

TCPUS is a set type that can contain multiple TCpu (920) values, enabling the specification of multiple target CPU architectures for a single build configuration. This allows packages and targets to support multiple CPU architectures simultaneously, facilitating cross-platform development and deployment strategies.

```
TDependencyType = (depPackage,depImplicitPackage,depUnit,depInclude
  )
```

Table 29.7: Enumeration values for type TDependencyType

Value	Explanation
depImplicitPackage	Implicit dependency on another package
depInclude	Dependency on an include file
depPackage	Dependency on another package
depUnit	Dependency on a specific unit

TDependencyType specifies the different kinds of dependencies that can be declared in the fpmake build system. Each type affects how the dependency is resolved and processed.

depPackage indicates a dependency on another complete package. This type ensures the referenced package is available and properly compiled before building the current package.

depImplicitPackage indicates an automatic dependency on another package that is discovered through code analysis. These dependencies are typically detected automatically without explicit declaration.

depUnit indicates a dependency on a specific Pascal unit. This type ensures the referenced unit is compiled and available before compiling units that use it.

depInclude indicates a dependency on an include file (.inc, .pas files) that is incorporated into source code using the {$I} directive. Ensures include files are available during compilation.

```
TDependencyTypes = Set of TDependencyType
```

A set type containing one or more values from the TDependencyType (921) enumeration. Used to specify multiple dependency types when filtering or processing package dependencies.

```
TDictionaryClass = Class of TDictionary
```

TDictionaryClass is a class reference type used for creating dictionary instances dynamically. This type enables runtime selection of different dictionary implementations and configurations.

```
TFileType = (ftSource,ftUnit,ftObject,ftResource,ftExecutable,
  ftStaticLibrary,ftSharedLibrary)
```

Table 29.8: Enumeration values for type TFileType

Value	Explanation
ftExecutable	Executable program file
ftObject	Object file produced by compilation
ftResource	Resource file containing application resources
ftSharedLibrary	Dynamic shared library file
ftSource	Pascal source code file
ftStaticLibrary	Static library file for linking
ftUnit	Compiled Pascal unit file

`TFileType` defines the different types of files that can be produced by build targets in the fpmake system. This classification determines how files are compiled, linked, and installed.

ftSource represents Pascal source code files (.pas, .pp, .p files) that contain program or unit source code. These files are compiled by the Free Pascal Compiler to produce executable programs or compiled units.

ftUnit represents compiled Pascal unit files (.ppu files) that contain compiled code and interface information. These files are produced by the compiler and used for linking with other units and programs.

ftObject represents object files (.o files on Unix, .obj on Windows) containing compiled machine code. These intermediate files are produced during compilation and used by the linker to create final executables or libraries.

ftResource represents resource files (.res, .rc files) containing application resources such as icons, bitmaps, strings, and other data. These files are compiled into applications to provide user interface elements and embedded data.

ftExecutable represents executable program files (.exe on Windows, no extension on Unix) that contain runnable machine code. These files are the final output of compiling and linking Pascal programs.

ftStaticLibrary represents static library files (.a files on Unix, .lib on Windows) containing compiled code that is linked directly into executables at compile time. Static libraries become part of the final executable.

ftSharedLibrary represents dynamic shared library files (.so on Unix, .dll on Windows) containing compiled code that is loaded at runtime. Shared libraries can be used by multiple programs simultaneously and are loaded dynamically when needed.

```
TFileTypes = Set of TFileType
```

`TFileTypes` is a set type that can contain multiple TFileType (921) values, allowing flexible specification of which file types are supported or required by build targets. This set-based approach enables targets to accept multiple input file types or produce different output types based on compilation settings.

```
TFPDocFormat = (ffHtml,ffHtm,ffXHtml,ffLaTex,ffXMLStruct,ffChm)
```

Table 29.9: Enumeration values for type TFPDocFormat

Value	Explanation
ffChm	Compressed HTML Help format
ffHtm	HTM format output
ffHtml	HTML format output
ffLaTex	LaTeX format output
ffXHtml	XHTML format output
ffXMLStruct	XML structured output

Enumeration defining the various output formats supported by the FPDoc documentation generator. Each format represents a different way to present the generated documentation.

ffHtml generates documentation in HTML format, creating web pages that can be viewed in any web browser.

ffHtm generates documentation in HTM format, an alternative HTML format with .htm file extension. This format provides the same functionality as HTML but uses the traditional 8.3 filename convention.

ffXHtml generates documentation in XHTML format, a stricter XML-compliant version of HTML.

ffLaTex generates documentation in LaTeX format, suitable for producing high-quality printed documentation. LaTeX output can be processed to create PDF, PostScript, or DVI files with professional typesetting.

ffXMLStruct generates documentation in structured XML format, providing a machine-readable representation of the documentation data.

ffChm generates documentation in Compressed HTML Help (CHM) format, creating Windows Help files with full-text search capabilities.

```
TFPDocFormats = Set of TFPDocFormat
```

A set type containing one or more values from the TFPDocFormat (922) enumeration. Allows specifying multiple documentation output formats to be generated simultaneously.

```
TfpmPluginClass = Class of TfpmPlugin
```

`TfpmPluginClass` is a class reference type used for registering and instantiating plugin classes. This type enables dynamic plugin creation and management within the plugin system.

```
TInstallerClass = Class of TCustomInstaller
```

`TInstallerClass` is a class reference type used for dynamic installer creation and management. This type enables runtime selection and instantiation of different installer implementations.

The class reference allows the system to create a custom installer instance.

```
TInstallMOde = (imInstall,imUnInstall)
```

Table 29.10: Enumeration values for type TInstallMOde

Value	Explanation
imInstall	Install package files
imUnInstall	Uninstall package files

`TInstallMode` specifies whether the current install run is installing or uninstalling.

imInstall specifies that the installation operation should install package files to their designated target locations in the system. This mode copies compiled units, executables, and resources to the appropriate installation directories.

imUnInstall specifies that the installation operation should remove previously installed package files from the system. This mode cleans up installed units, executables, and resources from the installation directories.

```
TLogEvent = procedure(Level: TVerboseLevel; const Msg: string) of
  object
```

Procedure type used for handling logging messages in the build system. Implemented by applications to receive and process log messages with specified verbosity levels.

```
TNotifyEventAction = (neaBeforeCompile,neaAfterCompile,neaBeforeInstall
  ,
                      neaAfterInstall,neaBeforeClean,neaAfterClean
  ,
                      neaBeforeArchive,neaAfterArchive,
  neaBeforeManifest,neaAfterManifest,
  neaBeforePkgList,neaAfterPkgList,
                      neaBeforeUnInstall
  ,neaAfterUnInstall,
                      neaBeforeCreateBuildEngine
  ,
                      neaAfterCreateBuildengine,neaBeforeDocProject
  ,
                      neaAfterDocProject)
```

Table 29.11: Enumeration values for type TNotifyEventAction

Value	Explanation
neaAfterArchive	Event triggered after archiving completes
neaAfterClean	Event triggered after cleanup completes
neaAfterCompile	Event triggered after compilation completes
neaAfterCreateBuildengine	Event triggered after build engine creation
neaAfterDocProject	Event triggered after documentation project processing
neaAfterInstall	Event triggered after installation completes
neaAfterManifest	Event triggered after manifest creation
neaAfterPkgList	Event triggered after package list creation
neaAfterUnInstall	Event triggered after uninstallation completes
neaBeforeArchive	Event triggered before archiving starts
neaBeforeClean	Event triggered before cleanup starts
neaBeforeCompile	Event triggered before compilation starts
neaBeforeCreateBuildEngine	Event triggered before build engine creation
neaBeforeDocProject	Event triggered before documentation project processing
neaBeforeInstall	Event triggered before installation starts
neaBeforeManifest	Event triggered before manifest creation
neaBeforePkgList	Event triggered before package list creation
neaBeforeUnInstall	Event triggered before uninstallation starts

`TNotifyEventAction` defines the different stages in the build process where notification events can be triggered.

This enumeration enables attaching custom actions or handlers to specific points in the build pipeline.

Build systems use these action points to execute custom commands, perform validation, or provide progress feedback.

```
TNotifyEventActionSet = Set of TNotifyEventAction
```

`TNotifyEventActionSet` is a set type that can contain multiple TNotifyEventAction (924) values.

This set enables efficient combination and testing of multiple event action types in a single variable.

Build systems use this set type to specify which events should trigger specific notification handlers.

```
TNotifyProcEvent = procedure(Sender: TObject)
```

Procedure type used for simple notification callbacks in the build system. Provides a sender parameter to identify the object that triggered the notification.

```
TOS = (osNone,linux,go32v2,win32,os2,freebsd,beos,netbsd,amiga,atari
  ,
       solaris,qnx,netware,openbsd,wdosx,palmos,macosclassic,darwin
  ,emx,
       watcom,morphos,netwlibc,win64,wince,gba,nds,embedded
  ,symbian,
       haiku,iphonesim,aix,java,android,nativent,msdos,wii
  ,aros,
       dragonfly,win16,ios)
```

Table 29.12: Enumeration values for type TOS

Value	Explanation
aix	IBM AIX operating system
amiga	Amiga operating system
android	Android mobile operating system
aros	AROS Research Operating System
atari	Atari TOS operating system
beos	BeOS operating system
darwin	macOS (Darwin) operating system
dragonfly	DragonFly BSD operating system
embedded	Generic embedded system
emx	EMX DOS/OS2 extender
freebsd	FreeBSD operating system
gba	Nintendo Game Boy Advance
go32v2	DOS 32-bit extender operating system
haiku	Haiku operating system
ios	Apple iOS mobile operating system
iphonesim	iOS Simulator environment
java	Java Virtual Machine platform
linux	Linux operating system
macosclassic	Classic Mac OS operating system
morphos	MorphOS operating system
msdos	MS-DOS operating system
nativent	Windows Native NT
nds	Nintendo DS handheld system
netbsd	NetBSD operating system
netware	Novell NetWare operating system
netwlibc	NetWare with LibC
openbsd	OpenBSD operating system
os2	IBM OS/2 operating system
osNone	No specific operating system
palmos	Palm OS operating system
qnx	QNX real-time operating system
solaris	Solaris operating system
symbian	Symbian mobile operating system
watcom	Watcom DOS extender
wdosx	WDOSX DOS extender
wii	Nintendo Wii gaming console
win16	Windows 16-bit operating system
win32	Windows 32-bit operating system
win64	Windows 64-bit operating system
wince	Windows CE operating system

`TOS` enumerates the operating systems supported by the Free Pascal Compiler for cross-platform compilation. This enumeration is used throughout the build system to specify target operating systems and handle OS-specific compilation and installation requirements.

```
TOSes = Set of TOS
```

`TOSes` is a set type that can contain multiple TOS (926) values, allowing the specification of multiple target operating systems for build configurations. This enables packages to define which operating

systems they support or require, and allows for flexible cross-platform compilation strategies within the FPMake system.

```
TPackageDictionaryClass = Class of TPackageDictionary
```

`TPackageDictionaryClass` is a class reference type specifically for package-related dictionary implementations. This type enables dynamic creation of dictionaries optimized for package configuration and metadata management.

```
TProcessPackageResult = (ppHandled,ppDelayed)
```

Table 29.13: Enumeration values for type TProcessPackageResult

Value	Explanation
ppDelayed	Package processing was delayed for later execution
ppHandled	Package was successfully processed and handled

`TProcessPackageResult` indicates the outcome of package processing operations.

Used to communicate success, failure, or special handling conditions during package operations.

ppHandled indicates that the package processing operation completed successfully and all required actions were performed. This result signals successful completion of the package operation.

ppDelayed indicates that the package processing operation has been delayed and will be executed at a later time. This result is used for dependency ordering or deferred processing scenarios.

```
TReplaceFunction = function(const AName: string; const Args: string
  )
                               : string of object
```

`TReplaceFunction` is a function pointer type for implementing dynamic string replacement callbacks.

Functions of this type are used in string substitution systems to provide custom replacement logic for variable expansion.

The function receives a name and arguments array, returning the replacement string value.

```
TRunMode = (rmCompile,rmBuild,rmInstall,rmBuildInstall,rmArchive,
  rmClean,rmDistClean,rmManifest,rmZipInstall,rmPkgList
  ,
            rmUnInstall,rmInfo,rmDocProject)
```

Table 29.14: Enumeration values for type TRunMode

Value	Explanation
rmArchive	Archive mode - create distribution archive
rmBuild	Build mode - compile and link
rmBuildInstall	Build and install mode
rmClean	Clean mode - remove generated files
rmCompile	Compile mode - only compile sources
rmDistClean	Distribution clean mode - remove all generated files and directories
rmDocProject	Generate documentation project file
rmInfo	Information mode - display package information
rmInstall	Install mode - install files
rmManifest	Generate package manifest file
rmPkgList	List available packages
rmUnInstall	Uninstall mode - remove installed package files
rmZipInstall	Install package from ZIP archive

Enumeration that defines the different run (operation) modes available in the build system. Determines what actions are performed when the build system is executed.

rmCompile specifies that the build system should only compile source files without performing installation or packaging operations. This mode produces compiled units and executables in the build directory.

rmBuild specifies that the build system should compile source files and perform linking operations to create final executables and libraries. This mode produces complete build artifacts ready for installation.

rmInstall specifies that the build system should install compiled files to their target locations in the system. This mode performs both building and installation of package components.

rmBuildInstall specifies that the build system should perform both compilation and installation operations in a single step. This mode combines building and installing for convenient package deployment.

rmArchive specifies that the build system should create distribution archives (.zip, .tar) containing package sources and compiled files. This mode prepares packages for distribution and deployment.

rmClean specifies that the build system should remove generated files such as compiled units, object files, and executables. This mode cleans the build environment for a fresh compilation.

rmDistClean specifies that the build system should perform a thorough cleanup, removing all generated files, directories, and build artifacts. This mode returns the package to its original source-only state.

rmManifest specifies that the build system should generate a package manifest file listing all package components, dependencies, and metadata. This mode creates documentation of package contents.

rmZipInstall specifies that the build system should install a package from a ZIP archive. This mode extracts and installs pre-packaged distribution files.

rmPkgList specifies that the build system should display a list of available packages in the system. This mode provides information about installable and installed packages.

rmUnInstall specifies that the build system should remove previously installed package files from the system. This mode performs cleanup of installed package components.

rmDocProject specifies that the build system should generate documentation project files for use with FPDoc or other documentation tools. This mode creates configuration files for documentation generation.

rmInfo specifies that the build system should display detailed information about a package, including version, dependencies, targets, and configuration.

```
TSourceType = (stDoc,stSrc,stExample,stTest)
```

Table 29.15: Enumeration values for type TSourceType

Value	Explanation
stDoc	Documentation source file
stExample	Example source file
stSrc	Pascal source code file
stTest	Test source file

Enumeration that classifies source files according to their type and role in the compilation process. Determines how each source file should be processed during building.

stDoc represents documentation source files, typically XML files used by FPDoc to generate formatted documentation. These files are processed separately from regular source code compilation.

stSrc represents Pascal source code files (.pas, .pp) that contain program or unit implementations. These files undergo normal compilation processing by the Free Pascal Compiler.

stTest represents test source files that contain unit tests or test programs for validating package functionality. These files are compiled and executed as part of the testing process.

stExample represents example or demonstration source files that showcase package functionality. These files are typically compiled separately and may have different installation rules than regular package sources.

```
TSourceTypes = Set of TSourceType
```

A set type containing one or more values from the TSourceType (929) enumeration. Used to specify multiple source file types for filtering or processing operations.

```
TTargetState = (tsNeutral,tsConsidering,tsNoCompile,tsCompiled,
  tsInstalled,tsNotFound)
```

Table 29.16: Enumeration values for type TTargetState

Value	Explanation
tsCompiled	Successfully compiled
tsConsidering	Being considered for compilation
tsInstalled	Successfully installed
tsNeutral	Neutral state, not yet processed
tsNoCompile	Should not be compiled
tsNotFound	Target file not found

`TTargetState` is an enumeration type that tracks the current state of a build target during the compilation process. Used to prevent duplicate processing and ensure correct build order dependencies. It has the following values:

tsNeutral indicates that a build target has not yet been processed or evaluated for compilation. This is the initial state of targets before build operations begin.

tsConsidering indicates that a build target is currently being evaluated for compilation. This intermediate state prevents infinite recursion during dependency analysis.

tsNoCompile indicates that a build target should be excluded from compilation. This state is used for targets that are not applicable to the current build configuration or platform.

tsCompiled indicates that a build target has been successfully compiled. Targets in this state have completed the compilation phase and are ready for installation or further processing.

tsInstalled indicates that a build target has been successfully installed to its destination location. This represents the final successful state in the build and installation process.

tsNotFound indicates that the source file for a build target could not be located. This error state prevents compilation and signals missing or incorrectly specified source files.

```
TTargetStates = Set of TTargetState
```

A set type containing one or more values from the TTargetState (929) enumeration. Used for filtering or checking multiple target states in batch operations.

```
TTargetType = (ttProgram,ttUnit,ttImplicitUnit,ttCleanOnlyUnit,
  ttExampleUnit,ttExampleProgram,ttFPDoc,ttSharedLibrary
  )
```

Table 29.17: Enumeration values for type TTargetType

Value	Explanation
ttCleanOnlyUnit	Unit processed only during clean operations
ttExampleProgram	Example program for demonstration purposes
ttExampleUnit	Example unit for demonstration purposes
ttFPDoc	FPDoc documentation generation target
ttImplicitUnit	Unit with implicit compilation dependencies
ttProgram	Executable program target
ttSharedLibrary	Dynamic shared library target
ttUnit	Regular Pascal unit target

`TTargetType` defines the different categories of build targets supported by the fpmake system:

ttProgram represents a build target that produces an executable program file

ttUnit represents a build target that produces a compiled Pascal unit. Unit targets generate .ppu files that can be used by other units and programs through the uses clause.

ttImplicitUnit represents a unit target that is compiled automatically when needed by other targets, without being explicitly listed in the build process.

ttCleanOnlyUnit represents a unit target that is only processed during clean operations to remove generated files. These units are not compiled during normal build processes but are included in cleanup operations.

ttExampleUnit represents a unit target that serves as an example or demonstration. Example units are typically compiled separately from the main package and may have different installation rules.

ttExampleProgram represents a program target that serves as an example or demonstration. Example programs showcase package functionality and are typically compiled separately from the main package components.

ttSharedLibrary represents a build target that produces a dynamic shared library (.so on Unix, .dll on Windows).

ttFPDoc represents a target that generates documentation using the FPDoc system. These targets documentation source code and XML documentation files.

```
TTargetTypes = Set of TTargetType
```

A set type containing one or more values from the TTargetType (930) enumeration.

```
TVerboseLevel = (vlError,vlWarning,vlInfo,vldebug,vlCommand)
```

Table 29.18: Enumeration values for type TVerboseLevel

Value	Explanation
vlCommand	Display executed commands
vldebug	Display debug messages
vlError	Display only error messages
vlInfo	Display informational messages
vlWarning	Display warning and error messages

Enumeration that controls the amount of information displayed during build operations. Higher verbosity levels include all messages from lower levels plus additional detail.

vlError specifies the lowest verbosity level, showing only critical error messages. This level provides minimal output, displaying only messages that indicate serious problems or failures.

vlWarning displays warning messages. This level shows potential problems and issues that may not prevent compilation but could affect the build process.

vlInfo displays general informational messages about the build process, including warnings and errors. This level provides feedback about what operations are being performed during compilation and installation.

vldebug displays detailed debug information for troubleshooting build issues. This level provides extensive output including internal operations, variable values, and detailed execution flow information.

vlCommand displays the actual commands being executed during the build process. This level shows compiler invocations, tool executions, and system commands, enabling verification and debugging of build operations.

```
TVerboseLevels = Set of TVerboseLevel
```

A set type containing one or more values from the TVerboseLevel (931) enumeration. Allows combining multiple verbosity levels for custom output filtering.

29.6.3 Variables

```
ArchiveFilesProc : TArchiveProc = Nil
```

`ArchiveFilesProc` is a global procedure variable that holds the archive implementation function. This variable enables system-wide configuration of the archive creation mechanism.

The archive procedure is called to perform actual file compression and archive creation, allowing pluggable archive implementations such as ZIP, TAR, or other compression formats.

```
Defaults : TCustomDefaults
```

`Defaults` is a global variable that holds the system-wide default configuration instance. This variable provides access to default build settings, paths, and options throughout the fpmake system.

The defaults instance contains configuration values for compiler paths, installation directories, build options, and other system-wide settings that apply to all package operations.

```
DictionaryClass : TDictionaryClass = TDictionary
```

`DictionaryClass` is a global variable that holds the class reference for the default dictionary implementation. This variable enables runtime configuration of which dictionary class to use throughout the build system.

The dictionary class is used for creating dictionary instances for template processing, variable substitution, and configuration management within fpmake operations.

```
GlobalDictionary : TDictionary
```

`GlobalDictionary` is a global dictionary instance used for system-wide variable storage and template processing. This dictionary provides a shared namespace for global configuration variables and template substitutions.

The global dictionary is accessible throughout the build system and enables sharing of common variables and settings across different packages and build operations.

```
OnArchiveFiles : TArchiveEvent = Nil
```

`OnArchiveFiles` is a global event handler variable for customizing archive file operations. This event handler enables system-wide customization of archive processing behavior.

When assigned, this event handler is called during archive operations to enable custom processing, validation, or modification of archive creation and file packaging operations.

```
PackageDictionaryClass : TPackageDictionaryClass =
  TPackageDictionary
```

PackageDictionaryClass is a global variable that holds the class reference for the default package dictionary implementation. This variable configures which dictionary class to use for package-specific operations.

The package dictionary class is used for creating specialized dictionaries that handle package metadata, configuration variables, and package-specific template processing within the build system.

29.7 Procedures and functions

29.7.1 AddCustomFpmakeCommandlineOption

Synopsis: Adds custom command line option to FPMake

Declaration:
```
procedure AddCustomFpmakeCommandlineOption
                                        (const ACommandLineOption: string;
                                        const HelpMessage: string)
```

Visibility: default

Description: Registers a custom command line option that will be recognized by the FPMake build system. This allows packages to define their own command line parameters for build customization.

After the command-line has been parsed, the value specified by the user can be retrieved with the GetCustomFpmakeCommandlineOptionValue (935) function.

See also: GetCustomFpmakeCommandlineOptionValue (935)

29.7.2 AddLibraryExtension

Synopsis: Adds appropriate extension to library name

Declaration:
```
function AddLibraryExtension(const LibraryName: string; AOS: TOS)
                                    : string
```

Visibility: default

Description: AddLibraryExtension adds the platform-appropriate file extension to a library name. This function ensures proper library naming conventions for different operating systems.

The function appends extensions such as .dll for Windows, .so for Linux, or .dylib for macOS, ensuring correct library file naming for the target platform.

29.7.3 AddProgramExtension

Synopsis: Adds appropriate extension to program name

Declaration:
```
function AddProgramExtension(const ExecutableName: string; AOS: TOS)
                                    : string
```

Visibility: default

Description: AddProgramExtension adds the platform-appropriate file extension to an executable program name. This function ensures proper executable naming conventions for different operating systems.

The function appends extensions such as .exe for Windows or leaves names unchanged for Unix-like systems, ensuring correct executable file naming for the target platform.

29.7.4 ChangeDir

Synopsis: Changes current working directory

Declaration: `procedure ChangeDir(const APath: string)`

Visibility: default

Description: `ChangeDir` changes the current working directory to the specified path and handles errors appropriately. This procedure provides controlled directory navigation during build operations.

The procedure validates the target directory exists and is accessible before changing the working directory.

29.7.5 CPUSToString

Synopsis: Converts set of CPU architectures to string

Declaration: `function CPUSToString(CPUS: TCPUS) : string`

Visibility: default

Description: `CPUSToString` converts a set of CPU architecture identifiers to a comma-separated string representation. This function provides a readable format for displaying multiple CPU architecture targets.

The string representation is used when logging supported architectures and configuration display.

29.7.6 CPUToString

Synopsis: Converts CPU architecture identifier to string

Declaration: `function CPUToString(CPU: TCpu) : string`

Visibility: default

Description: `CPUToString` converts a CPU architecture identifier enumeration value to its string representation. This function provides human-readable names for CPU architectures used in build configuration.

The string representation is useful for logging, configuration files, and user interface display of architecture-specific information within the build system.

29.7.7 CurrentCPU

Synopsis: Gets the current CPU architecture

Declaration: `function CurrentCPU : string`

Visibility: default

Description: `CurrentCPU` returns the current CPU architecture identifier for the running platform. This function provides architecture detection for conditional compilation and architecture-specific build settings.

The CPU architecture identifier is used to select appropriate compiler targets, optimization options, and platform-specific code generation settings during the build process.

29.7.8 CurrentOS

Synopsis: Gets the current operating system

Declaration:
```
function CurrentOS : string
```

Visibility: default

Description: `CurrentOS` returns the current operating system identifier for the running platform. This function provides platform detection for conditional compilation and platform-specific build operations.

The operating system identifier is used to select appropriate compiler options, file paths, and installation procedures for the target platform.

29.7.9 FixPath

Synopsis: Normalizes and fixes file path format

Declaration:
```
function FixPath(const APath: string) : string
function FixPath(const APath: string; AIsDir: Boolean) : string
```

Visibility: default

Description: `FixPath` normalizes file paths by converting directory separators to the platform-appropriate format and ensuring consistent path representation. This function handles cross-platform path compatibility.

The normalization includes converting forward/backward slashes, removing redundant separators, and ensuring proper directory path formatting for the target operating system.

29.7.10 GetCustomFpmakeCommandlineOptionValue

Synopsis: Gets value of custom command line option

Declaration:
```
function GetCustomFpmakeCommandlineOptionValue
                                                (const ACommandLineOption: string)
                                                 : string
```

Visibility: default

Description: `GetCustomFpmakeCommandlineOptionValue` retrieves the value of a previously registered custom command line option. This function enables packages to access their custom command-line parameters during build operations.

The function returns the value specified for the command line option.

The custom command-line option must have been registered using the AddCustomFpmakeCommandlineOption (933) function first.

See also: AddCustomFpmakeCommandlineOption (933)

29.7.11 GetDefaultLibGCCDir

Synopsis: Gets default GCC library directory

Declaration:
```
function GetDefaultLibGCCDir(CPU: TCpu; OS: TOS;
                             var ErrorMessage: string) : string
```

Visibility: default

Description: `GetDefaultLibGCCDir` determines the default GCC library directory for the specified CPU architecture and operating system. This function locates system GCC libraries required for linking compiled programs.

The function searches standard locations and returns the appropriate GCC library path.

29.7.12 GetImportLibraryFilename

Synopsis: Gets filename for import library

Declaration:
```
function GetImportLibraryFilename(const UnitName: string; AOS: TOS)
                                        : string
```

Visibility: default

Description: `GetImportLibraryFilename` generates the appropriate import library filename for a unit on the specified operating system. Import libraries are used on platforms like Windows to link against dynamic libraries.

The function constructs the correct import library name following platform conventions, enabling proper linking with dynamic libraries during the compilation process.

29.7.13 GetPluginManager

Synopsis: Gets the global plugin manager instance

Declaration:
```
function GetPluginManager : TfpmPluginManager
```

Visibility: default

Description: Returns the global plugin manager instance used by the FPMake build system. The plugin manager handles loading and managing build system extensions and add-ins.

29.7.14 Installer

Synopsis: Gets the current installer instance

Declaration:
```
function Installer(InstallerClass: TInstallerClass) : TCustomInstaller
                    ;  Overload
function Installer : TCustomInstaller;  Overload
```

Visibility: default

Description: `Installer` returns the current installer instance or creates a new one using the specified installer class. Note that only one class can be instantiated. This function provides access to the global installer object for package management operations.

The installer instance manages the build process, package compilation, installation, and other package lifecycle operations within the fpmake system.

If no class is specified and no instance is present yet, the TFPCInstaller (994) class is used to create the installer class.

See also: TFPCInstaller (994)

29.7.15 IsDifferentFromBuild

Synopsis: Checks if target differs from build platform

Declaration: `function IsDifferentFromBuild(ACpu: TCpu; AOs: TOS) : Boolean`

Visibility: default

Description: `IsDifferentFromBuild` determines whether the specified target operating system and CPU architecture differ from the current build platform. This function enables detection of cross-compilation scenarios.

The function returns `True` when building for a different platform than the current build system.

29.7.16 IsRelativePath

Synopsis: Checks if a path is relative

Declaration: `function IsRelativePath(const APath: string) : Boolean`

Visibility: default

Description: `IsRelativePath` determines whether a file path is relative or absolute by examining its format. This function provides platform-aware path type detection for build system operations.

The function returns `True` for relative paths and `False` for absolute paths.

29.7.17 MakeTargetString

Synopsis: Creates target string from parameters

Declaration: `function MakeTargetString(CPU: TCpu; OS: TOS) : string`

Visibility: default

Description: `MakeTargetString` constructs a formatted target string from CPU architecture, operating system, and build mode parameters. This function creates standardized target identifiers used throughout the build system.

The generated target string format enables consistent identification of build configurations and target platforms within package management and compilation operations.

29.7.18 ModeToString

Synopsis: Converts build mode to string representation

Declaration: `function ModeToString(Mode: TCompilerMode) : string`

Visibility: default

Description: `ModeToString` converts a build mode identifier enumeration to its string representation. This function provides human-readable names for build modes used in build configuration.

The string representation includes modes such as debug, release, and other build configurations useful for logging and user interface display within the build system.

29.7.19 OSesToString

Synopsis: Converts set of operating systems to string

Declaration:
```
function OSesToString(OSes: TOSes) : string
```

Visibility: default

Description: `OSesToString` converts a set of operating system identifiers to a comma-separated string representation. This function provides a readable format for displaying multiple operating system targets.

The string representation is useful for logging supported platforms, configuration display, and documentation of multi-platform package compatibility.

29.7.20 OSToString

Synopsis: Converts operating system identifier to string

Declaration:
```
function OSToString(OS: TOS) : string
```

Visibility: default

Description: `OSToString` converts an operating system identifier enumeration value to its string representation. This function provides human-readable names for operating system identifiers used in build configuration.

The string representation is useful for logging, configuration files, and user interface display of platform-specific information within the build system.

29.7.21 SearchFiles

Synopsis: Searches for files in specified paths

Declaration:
```
procedure SearchFiles(AFileName: string; ASearchPathPrefix: string;
                      Recursive: Boolean; var List: TStrings)
```

Visibility: default

Description: `SearchFiles` searches for files matching a filename pattern within specified search paths. This procedure supports recursive directory traversal and wildcard pattern matching for file discovery.

The procedure populates a provided list with all matching files found during the search, enabling file location and collection for build operations.

29.7.22 SplitCommand

Synopsis: Splits command line into executable and options

Declaration:
```
procedure SplitCommand(const Cmd: string; out Exe: string;
                       out Options: string)
```

Visibility: default

Description: `SplitCommand` parses a complete command line string and separates it into the executable name and its command-line options. This procedure handles quoted arguments and proper command-line parsing for the build system.

The procedure populates separate output parameters with the executable name and the remaining options.

29.7.23 StringToCPU

Synopsis: Converts string to CPU architecture identifier

Declaration: `function StringToCPU(const S: string) : TCpu`

Visibility: default

Description: `StringToCPU` converts a string representation of a CPU architecture to its corresponding identifier enumeration. This function enables parsing of CPU architecture names from configuration files and command-line parameters.

The function supports standard CPU architecture names.

29.7.24 StringToCPUOS

Synopsis: Parses string to extract CPU and OS information

Declaration: `procedure StringToCPUOS(const S: string; var CPU: TCpu; var OS: TOS)`

Visibility: default

Description: `StringToCPUOS` parses a combined string containing CPU architecture and operating system information, extracting the individual CPU and OS identifiers. This procedure handles target string parsing for build configuration.

The procedure processes target strings in various formats and populates the output parameters with the corresponding CPU architecture and operating system enumeration values.

29.7.25 StringToCPUS

Synopsis: Converts string to CPU architecture set

Declaration: `function StringToCPUS(const S: string) : TCPUS`

Visibility: default

Description: `StringToCPUS` converts a comma-separated string of CPU architectures to a set of CPU identifier enumerations. This function enables parsing of multiple CPU architecture targets from configuration strings.

The function processes comma-separated architecture names and builds a set containing all valid CPU architecture identifiers found in the input string.

29.7.26 StringToMode

Synopsis: Converts string to build mode identifier

Declaration: `function StringToMode(const S: string) : TCompilerMode`

Visibility: default

Description: `StringToMode` converts a string representation of a build mode to its corresponding identifier enumeration. This function enables parsing of build mode names from configuration files and command-line parameters.

The function supports standard build mode names such as debug, release, and custom modes, providing error handling for invalid or unrecognized mode strings.

29.7.27 StringToOS

Synopsis: Converts string to operating system identifier

Declaration: `function StringToOS(const S: string) : TOS`

Visibility: default

Description: `StringToOS` converts a string representation of an operating system to its corresponding identifier enumeration. This function enables parsing of operating system names from configuration files and command-line parameters.

The function supports standard operating system names.

29.8 ECollectionError

29.8.1 Description

`ECollectionError` is raised when errors occur during collection operations such as invalid indices, missing items, or collection manipulation failures.

29.9 EDictionaryError

29.9.1 Description

`EDictionaryError` is raised when dictionary operations fail due to invalid keys, missing entries, or other dictionary-specific error conditions.

29.10 EInstallerError

29.10.1 Description

`EInstallerError` is raised when installer operations encounter errors such as installation failures, permission problems, or configuration issues during package installation.

29.11 TBasicDefaults

29.11.1 Description

`TBasicDefaults` provides a concrete implementation of TCustomDefaults (967) with basic configuration management for the fpmake build system.

This class manages fundamental build settings including compiler paths, installation directories, archiving options, and other essential configuration parameters for package building and installation.

29.12 TBasicInstaller

29.12.1 Description

`TBasicInstaller` provides a basic implementation of TCustomInstaller (983) with standard functionality for package building and installation. This class offers a complete installer without compiler-specific optimizations.

The basic installer provides general-purpose package management suitable for the FPC compiler.

29.12.2 Method overview

Page	Method	Description
941	Create	Creates basic installer instance

29.12.3 TBasicInstaller.Create

Synopsis: Creates basic installer instance

Declaration: `constructor Create(AOwner: TComponent);  Override`

Visibility: default

Description: `Create` initializes a new `TBasicInstaller` instance with standard configuration and defaults. This constructor sets up the installer with general-purpose build engine and basic package management capabilities.

29.13 TBuildEngine

29.13.1 Description

`TBuildEngine` is the central component that orchestrates the entire build process for fpmake packages. It manages compilation, dependency resolution, file operations, and installation procedures.

The build engine provides the following functionality:

- Package compilation with dependency resolution
- Cross-platform build support
- File operations (copy, move, delete, archive)
- Installation and uninstallation procedures
- Clean and archive operations
- Multi-threaded compilation support
- Command execution and logging

The engine works with TPackage (1009) objects to compile targets, resolve dependencies, and manage the entire build lifecycle from compilation through installation.

See also: TPackage (1009), TTarget (1049), TCustomDefaults (967), TCustomInstaller (983)

29.13.2 Method overview

Page	Method	Description
943	AddPathPrefix	Adds package-specific path prefix to a path
952	Archive	Creates archive of package files
952	CheckDependencies	Checks package dependencies are satisfied
953	CheckExternalPackage	Checks if external package is available
952	Clean	Cleans compiled files and outputs
944	ClearResolvedFileNames	Clears resolved file name cache for a package
945	CmdArchiveFiles	Creates archive file containing specified files
944	CmdCopyFiles	Copies files to destination directory
944	CmdCreateDir	Creates a directory with necessary parent directories
945	CmdDeleteDestFiles	Deletes files from destination directory
945	CmdDeleteFiles	Deletes files from filesystem
945	CmdMoveFiles	Moves files to destination directory
946	CmdRemoveDirs	Removes directories from the filesystem
946	CmdRemoveTrees	Removes directory trees from the filesystem
946	CmdRenameFile	Renames or moves a file to new location
948	Compile	Compiles a specific target within a package
948	CompileDependencies	Compiles all dependencies for a target
943	Create	Creates a new build engine instance
953	CreateOutputDir	Creates output directory for package
946	DependencyOK	Checks if dependency is satisfied
943	Destroy	Destroys the build engine and frees resources
950	DoAfterArchive	Executes actions after package archiving
950	DoAfterClean	Executes actions after package cleaning
949	DoAfterCompile	Executes actions after package compilation
949	DoAfterInstall	Executes actions after package installation
950	DoBeforeArchive	Executes actions before package archiving
950	DoBeforeClean	Executes actions before package cleaning
949	DoBeforeCompile	Executes actions before package compilation
949	DoBeforeInstall	Executes actions before package installation
944	ExecuteCommand	Executes a command with arguments and environment variables
946	ExecuteCommands	Executes a collection of build commands
954	FPDocProject	Generates FPDoc documentation project
947	GetCompilerCommand	Gets compiler command for target
951	Install	Installs package files to target location
954	Log	Logs message with specified level and formatting
954	Manifest	Creates manifest of package contents
948	MaybeCompile	Compiles target if compilation is needed
947	NeedsCompile	Checks if target needs compilation
948	PackageOK	Checks if package is valid for compilation
952	PkgList	Lists package contents and information
951	ReadyToCompile	Checks if package is ready for compilation
943	ResolveFileNames	Resolves and validates file paths for package targets
947	TargetInstallOK	Checks if target can be installed on platform
947	TargetOK	Checks if target is valid for platform
951	UnInstall	Uninstalls package files from target location
953	ZipInstall	Creates ZIP archive during installation

29.13.3 Property overview

Page	Properties	Access	Description
955	ExternalPackages	r	Collection of external package dependencies
955	ForceCompile	rw	Whether to force recompilation of all targets
955	Interactive	rw	Whether the build engine runs in interactive mode
955	ListMode	rw	Mode for listing package information
956	NotifyEventCollection	r	Collection of notification event handlers
956	OnLog	rw	Event handler for logging messages
956	StartDir	r	Starting directory for build operations
954	Verbose	rw	Controls verbose output during build operations

29.13.4 TBuildEngine.Create

Synopsis: Creates a new build engine instance

Declaration:
```
constructor Create(AOwner: TComponent);  Override
```

Visibility: public

Description: Creates a new `TBuildEngine` instance and initializes all internal components needed for package compilation, installation, and management.

The build engine handles compilation processes, dependency resolution, file operations, and package installation coordination.

29.13.5 TBuildEngine.Destroy

Synopsis: Destroys the build engine and frees resources

Declaration:
```
destructor Destroy;  Override
```

Visibility: public

Description: Destroys the `TBuildEngine` instance and frees all associated resources including critical sections, archive handlers, and temporary files.

29.13.6 TBuildEngine.AddPathPrefix

Synopsis: Adds package-specific path prefix to a path

Declaration:
```
function AddPathPrefix(APackage: TPackage; APath: string) : string
```

Visibility: public

Description: Adds a package-specific path prefix to the given path string.

This ensures paths are correctly resolved relative to the package's base directory and build configuration.

29.13.7 TBuildEngine.ResolveFileNames

Synopsis: Resolves and validates file paths for package targets

Declaration:
```
procedure ResolveFileNames(APackage: TPackage; ACPU: TCpu; AOS: TOS;
                           DoChangeDir: Boolean; WarnIfNotFound: Boolean)
```

Visibility: public

Description: Resolves all file paths used by targets in the package, ensuring source files exist and creating proper search paths for the specified CPU and OS.

This critical preprocessing step validates that all required files are available before compilation begins and sets up directory context for build operations.

29.13.8 TBuildEngine.ClearResolvedFileNames

Synopsis: Clears resolved file name cache for a package

Declaration:
```
procedure ClearResolvedFileNames(APackage: TPackage)
```

Visibility: public

Description: Clears all cached resolved file names and paths for the specified package.

This is typically called after compilation or when package configuration changes to ensure fresh file resolution on subsequent operations.

29.13.9 TBuildEngine.ExecuteCommand

Synopsis: Executes a command with arguments and environment variables

Declaration:
```
procedure ExecuteCommand(const Cmd: string; const Args: string;
                         const Env: TStrings; IgnoreError: Boolean)
                         ;  Virtual
```

Visibility: public

Description: Executes an external command with the specified arguments and environment variables. The command execution can optionally ignore errors based on the `IgnoreError` parameter.

This is used by the build system to invoke external tools like compilers, linkers, archive utilities, and other build tools during the compilation and installation process.

29.13.10 TBuildEngine.CmdCopyFiles

Synopsis: Copies files to destination directory

Declaration:
```
procedure CmdCopyFiles(List: TStrings; const DestDir: string;
                       APackage: TPackage)
```

Visibility: public

Description: Copies a list of files to the specified destination directory using either the configured copy command or the system's native copy functionality.

The destination directory is created if it doesn't exist. File paths can be absolute or relative to the current working directory.

29.13.11 TBuildEngine.CmdCreateDir

Synopsis: Creates a directory with necessary parent directories

Declaration:
```
procedure CmdCreateDir(const DestDir: string)
```

Visibility: public

Description: Creates a directory and all necessary parent directories in the path. If the directory already exists, no error is generated.

This is used to ensure output directories exist before files are written during the build and installation process.

29.13.12 TBuildEngine.CmdMoveFiles

Synopsis: Moves files to destination directory

Declaration: `procedure CmdMoveFiles(List: TStrings; const DestDir: string)`

Visibility: public

Description: Moves a list of files to the specified destination directory using either the configured move command or the system's native move functionality.

Files are relocated from their current location to the destination directory. The destination directory is created if it doesn't exist.

29.13.13 TBuildEngine.CmdDeleteFiles

Synopsis: Deletes files from filesystem

Declaration: `procedure CmdDeleteFiles(List: TStrings)`

Visibility: public

Description: Deletes a list of files from the filesystem using either the configured remove command or the system's native delete functionality.

This is typically used during clean operations to remove compiled object files, executables, and other generated files.

29.13.14 TBuildEngine.CmdDeleteDestFiles

Synopsis: Deletes files from destination directory

Declaration: `procedure CmdDeleteDestFiles(List: TStrings; const DestDir: string)`

Visibility: public

Description: Deletes a list of files from the specified destination directory.

This method combines the file list with the destination directory path and removes the specified files as part of cleanup operations.

29.13.15 TBuildEngine.CmdArchiveFiles

Synopsis: Creates archive file containing specified files

Declaration: `procedure CmdArchiveFiles(List: TStrings; const ArchiveFile: string)`

Visibility: public

Description: Creates an archive file containing the specified list of files using either the configured archive command or the system's native archiving functionality.

This is used during the archive phase to create distribution packages containing compiled binaries and resource files.

29.13.16 TBuildEngine.CmdRenameFile

Synopsis: Renames or moves a file to new location

Declaration: `procedure CmdRenameFile(SourceName: string; DestName: string)`

Visibility: public

Description: Renames a file from the source location to the destination location. This can be used for both simple renaming within the same directory or moving files between different directories.

Used during build processes when files need to be relocated or renamed according to the package configuration.

29.13.17 TBuildEngine.CmdRemoveDirs

Synopsis: Removes directories from the filesystem

Declaration: `procedure CmdRemoveDirs(List: TStrings)`

Visibility: public

Description: Removes a list of empty directories from the filesystem using the configured directory removal command.

This method is typically used during cleanup operations to remove temporary and generated directories that are no longer needed.

29.13.18 TBuildEngine.CmdRemoveTrees

Synopsis: Removes directory trees from the filesystem

Declaration: `procedure CmdRemoveTrees(List: TStrings)`

Visibility: public

Description: Recursively removes entire directory trees including all subdirectories and files within them. This is used during clean operations to remove generated directory structures.

29.13.19 TBuildEngine.ExecuteCommands

Synopsis: Executes a collection of build commands

Declaration:
```
procedure ExecuteCommands(Commands: TCommands; At: TCommandAt;
                          APackage: TPackage)
```

Visibility: public

Description: Executes all commands in the specified collection at the given timing point during the build process. This allows custom commands to be run at specific phases of package building or installation.

29.13.20 TBuildEngine.DependencyOK

Synopsis: Checks if dependency is satisfied

Declaration: `function DependencyOK(ADependency: TDependency) : Boolean`

Visibility: public

Description: DependencyOK verifies whether a specified package dependency is satisfied and available for the build process. This function checks dependency resolution, version requirements, and availability of the required package.

The method returns True if the dependency is properly resolved and can be used during compilation, or False if the dependency is missing, has version conflicts, or cannot be satisfied.

29.13.21 TBuildEngine.GetCompilerCommand

Synopsis: Gets compiler command for target

Declaration:
```
function GetCompilerCommand(APackage: TPackage; ATarget: TTarget;
                            Env: TStrings) : string
```

Visibility: public

Description: Constructs the complete compiler command line for compiling the specified target.

The command includes the compiler executable, all necessary options, paths, and parameters required to compile the target successfully.

29.13.22 TBuildEngine.TargetOK

Synopsis: Checks if target is valid for platform

Declaration:
```
function TargetOK(ATarget: TTarget; ACPU: TCpu; AOS: TOS) : Boolean
```

Visibility: public

Description: Determines whether the specified target is valid for the current platform configuration.

This check considers the target's CPU and OS restrictions to determine if it should be built on the current system.

29.13.23 TBuildEngine.TargetInstallOK

Synopsis: Checks if target can be installed on platform

Declaration:
```
function TargetInstallOK(ATarget: TTarget; ACPU: TCpu; AOS: TOS)
                         : Boolean
```

Visibility: public

Description: TargetInstallOK determines whether a build target can be installed on the current platform and configuration. This function evaluates platform compatibility, dependency availability, and installation requirements.

The method returns True if the target is suitable for installation on the current system, or False if platform restrictions or missing dependencies prevent installation.

29.13.24 TBuildEngine.NeedsCompile

Synopsis: Checks if target needs compilation

Declaration:
```
function NeedsCompile(APackage: TPackage; ATarget: TTarget) : Boolean
function NeedsCompile(APackage: TPackage) : Boolean;  Virtual
```

Visibility: public

Description: Determines whether the specified target needs to be compiled by checking timestamps, dependencies, and build requirements.

Returns `True` if the target's output files are missing, out of date, or if dependencies have changed.

29.13.25 TBuildEngine.Compile

Synopsis: Compiles a specific target within a package

Declaration:
```
procedure Compile(APackage: TPackage; ATarget: TTarget);  Virtual
procedure Compile(APackage: TPackage)
procedure Compile(Packages: TPackages)
```

Visibility: public

Description: Compiles the specified target within the given package. This method handles the actual compilation process, including dependency resolution, command-line generation, and execution of the Free Pascal Compiler.

See also: MaybeCompile (948), CompileDependencies (948), TTarget (1049), TPackage (1009)

29.13.26 TBuildEngine.MaybeCompile

Synopsis: Compiles target if compilation is needed

Declaration:
```
procedure MaybeCompile(APackage: TPackage; ATarget: TTarget)
procedure MaybeCompile(APackage: TPackage)
```

Visibility: public

Description: Conditionally compiles the target only if compilation is required based on dependency analysis and timestamp checks.

This method first calls NeedsCompile (947) and only performs compilation if necessary.

29.13.27 TBuildEngine.CompileDependencies

Synopsis: Compiles all dependencies for a target

Declaration:
```
procedure CompileDependencies(APackage: TPackage; ATarget: TTarget)
procedure CompileDependencies(APackage: TPackage)
```

Visibility: public

Description: Ensures that all dependencies for the specified target are compiled before compiling the target itself. This method recursively processes the dependency chain to maintain proper build order.

See also: Compile (948), TDependency (988), TTarget (1049)

29.13.28 TBuildEngine.PackageOK

Synopsis: Checks if package is valid for compilation

Declaration:
```
function PackageOK(APackage: TPackage) : Boolean;  Virtual
```

Visibility: public

Description: `PackageOK` validates whether a package is properly configured and ready for compilation. This function examines package structure, dependencies, target definitions, and configuration validity.

The method returns `True` if the package can be compiled successfully, or `False` if there are configuration errors, missing dependencies, or other issues preventing compilation.

29.13.29 TBuildEngine.DoBeforeCompile

Synopsis: Executes actions before package compilation

Declaration: `procedure DoBeforeCompile(APackage: TPackage); Virtual`

Visibility: public

Description: `DoBeforeCompile` performs pre-compilation setup and initialization tasks for the package. This method prepares the build environment, resolves dependencies, and executes any custom pre-build commands.

The procedure ensures that all necessary conditions are met before the actual compilation process begins, including directory creation, file preparation, and environment configuration.

29.13.30 TBuildEngine.DoAfterCompile

Synopsis: Executes actions after package compilation

Declaration: `procedure DoAfterCompile(APackage: TPackage); Virtual`

Visibility: public

Description: `DoAfterCompile` performs post-compilation cleanup and finalization tasks for the package. This method executes custom post-build commands and performs any necessary housekeeping after compilation completes.

The procedure handles result validation, temporary file cleanup, and any additional processing required after the compilation phase has finished successfully.

29.13.31 TBuildEngine.DoBeforeInstall

Synopsis: Executes actions before package installation

Declaration: `procedure DoBeforeInstall(APackage: TPackage); Virtual`

Visibility: public

Description: `DoBeforeInstall` prepares the system and package for installation by executing pre-installation tasks. This method ensures installation directories exist, validates installation requirements, and runs custom pre-install commands.

The procedure verifies installation permissions, creates necessary directories, and performs any setup required before the actual file installation begins.

29.13.32 TBuildEngine.DoAfterInstall

Synopsis: Executes actions after package installation

Declaration: `procedure DoAfterInstall(APackage: TPackage); Virtual`

Visibility: public

Description: `DoAfterInstall` completes the installation process by executing post-installation tasks. This method runs custom post-install scripts, registers the package, and performs final installation verification.

The procedure handles package registration, configuration updates, and any cleanup or finalization steps required after successful package installation.

29.13.33 TBuildEngine.DoBeforeArchive

Synopsis: Executes actions before package archiving

Declaration: `procedure DoBeforeArchive(APackage: TPackage); Virtual`

Visibility: public

Description: `DoBeforeArchive` prepares files and directories for package archive creation. This method organizes files, validates archive contents, and executes pre-archiving customization commands.

The procedure ensures all necessary files are included, applies any transformations, and sets up the archive structure before the compression process begins.

29.13.34 TBuildEngine.DoAfterArchive

Synopsis: Executes actions after package archiving

Declaration: `procedure DoAfterArchive(APackage: TPackage); Virtual`

Visibility: public

Description: `DoAfterArchive` completes the archiving process by executing post-archive tasks. This method validates the created archive, runs post-archiving scripts, and performs cleanup operations.

The procedure verifies archive integrity, moves files to final locations, and handles any additional processing required after successful archive creation.

29.13.35 TBuildEngine.DoBeforeClean

Synopsis: Executes actions before package cleaning

Declaration: `procedure DoBeforeClean(APackage: TPackage); Virtual`

Visibility: public

Description: `DoBeforeClean` prepares for package cleaning by executing pre-cleanup tasks. This method backs up important files, runs custom pre-clean scripts, and identifies files to be removed.

The procedure ensures that critical files are preserved and performs any necessary preparation before the cleaning process removes temporary and generated files.

29.13.36 TBuildEngine.DoAfterClean

Synopsis: Executes actions after package cleaning

Declaration: `procedure DoAfterClean(APackage: TPackage); Virtual`

Visibility: public

Description: `DoAfterClean` completes the cleaning process by executing post-cleanup tasks. This method validates that unwanted files were removed, runs post-clean verification, and performs final cleanup steps.

The procedure ensures the cleaning was successful and handles any additional housekeeping required after the package cleanup operation completes.

29.13.37 TBuildEngine.ReadyToCompile

Synopsis: Checks if package is ready for compilation

Declaration:
```
function ReadyToCompile(APackage: TPackage) : Boolean
```

Visibility: public

Description: `ReadyToCompile` verifies that all prerequisites are met for package compilation. This function checks dependency availability, compiler settings, source file presence, and build configuration validity.

The method returns `True` if the package can be compiled successfully, or `False` if missing dependencies, configuration errors, or other issues prevent compilation.

29.13.38 TBuildEngine.Install

Synopsis: Installs package files to target location

Declaration:
```
procedure Install(APackage: TPackage; AnArchiveFiles: Boolean)
procedure Install(Packages: TPackages)
```

Visibility: public

Description: `Install` performs the complete package installation process by copying compiled files and resources to their target installation directories. This method handles library files, executables, documentation, and other package components.

The installation process includes directory creation, file copying with proper permissions, and optional archive creation for backup and distribution purposes.

29.13.39 TBuildEngine.UnInstall

Synopsis: Uninstalls package files from target location

Declaration:
```
procedure UnInstall(APackage: TPackage)
procedure UnInstall(Packages: TPackages)
```

Visibility: public

Description: `UnInstall` removes package files from their installation directories and performs cleanup. This method reverses the installation process by deleting installed files, libraries, and associated resources.

The uninstallation includes removing executable files, libraries, documentation, and configuration files while preserving user data and custom configurations where appropriate.

29.13.40 TBuildEngine.Archive

Synopsis: Creates archive of package files

Declaration:
```
procedure Archive(APackage: TPackage)
procedure Archive(Packages: TPackages)
```

Visibility: public

Description: `Archive` creates compressed archive files containing package sources, binaries, and documentation. This method generates distributable archive files in formats such as ZIP, TAR, or other compression formats.

The archiving process includes source files, compiled binaries, documentation, examples, and other package components organized for distribution and installation.

29.13.41 TBuildEngine.PkgList

Synopsis: Lists package contents and information

Declaration:
```
procedure PkgList(PkgList: TStrings; APackage: TPackage)
procedure PkgList(Packages: TPackages)
```

Visibility: public

Description: `PkgList` displays detailed information about package contents, dependencies, targets, and configuration. This method provides a listing of package components for review and verification.

The listing includes source files, dependencies, build targets, installation paths, and other package metadata useful for package management and debugging.

29.13.42 TBuildEngine.Clean

Synopsis: Cleans compiled files and outputs

Declaration:
```
procedure Clean(APackage: TPackage; AllTargets: Boolean)
procedure Clean(APackage: TPackage; ACPU: TCpu; AOS: TOS)
procedure Clean(Packages: TPackages; AllTargets: Boolean)
```

Visibility: public

Description: `Clean` removes compiled object files, executables, and other generated outputs from the build process. This method cleans up temporary files, object files, and build artifacts to prepare for a fresh compilation.

The cleaning process can target specific packages or all targets, removing compiled units, executables, and temporary files while preserving source code and configuration files.

29.13.43 TBuildEngine.CheckDependencies

Synopsis: Checks package dependencies are satisfied

Declaration:
```
function CheckDependencies(APackage: TPackage; ErrorOnFailure: Boolean)
                                  : TCheckDependencyResult
```

Visibility: public

Description: `CheckDependencies` validates that all required dependencies for a package are available and properly configured. This function verifies dependency versions, availability, and compatibility with the current build configuration.

The method returns `True` if all dependencies are satisfied, or `False` if missing dependencies or version conflicts prevent successful compilation.

29.13.44 TBuildEngine.CheckExternalPackage

Synopsis: Checks if external package is available

Declaration:
```
function CheckExternalPackage(const APackageName: string;
                              const ForPackageName: string;
                              ErrorOnFailure: Boolean) : TPackage
```

Visibility: public

Description: `CheckExternalPackage` verifies the availability and accessibility of external packages required by the build. This function checks for system packages, third-party libraries, and external dependencies not managed by fpmake.

The method returns `True` if the external package is found and accessible, enabling the build to proceed with confidence that external dependencies are satisfied.

29.13.45 TBuildEngine.CreateOutputDir

Synopsis: Creates output directory for package

Declaration:
```
procedure CreateOutputDir(APackage: TPackage)
```

Visibility: public

Description: `CreateOutputDir` creates the necessary output directory structure for package compilation and installation. This method ensures that target directories exist with proper permissions for storing compiled files and outputs.

The procedure creates unit output directories, executable paths, and library directories as required by the package configuration and target platform.

29.13.46 TBuildEngine.ZipInstall

Synopsis: Creates ZIP archive during installation

Declaration:
```
procedure ZipInstall(Packages: TPackages)
```

Visibility: public

Description: `ZipInstall` combines package installation with ZIP archive creation for convenient distribution. This method installs package files to their target locations while simultaneously creating a compressed ZIP archive.

The ZIP installation process enables easy package distribution and backup by creating both installed files and a portable archive containing all package components.

29.13.47 TBuildEngine.Manifest

Synopsis: Creates manifest of package contents

Declaration:
```
procedure Manifest(Packages: TPackages; Package: TPackage)
```

Visibility: public

Description: `Manifest` generates manifest files containing detailed package information and contents. This method creates structured documentation of package files, dependencies, versions, and configuration.

The manifest includes file listings, checksums, dependency information, and metadata useful for package management, verification, and deployment processes.

29.13.48 TBuildEngine.FPDocProject

Synopsis: Generates FPDoc documentation project

Declaration:
```
procedure FPDocProject(Packages: TPackages; SingleDocFile: Boolean)
```

Visibility: public

Description: `FPDocProject` creates FPDoc documentation projects for generating API documentation. This method processes package source files and generates structured documentation using the FPDoc tool.

The documentation project includes unit descriptions, class hierarchies, method signatures, and code examples organized for web-based or printed documentation distribution.

29.13.49 TBuildEngine.Log

Synopsis: Logs message with specified level and formatting

Declaration:
```
procedure Log(Level: TVerboseLevel; Msg: string)
procedure Log(Level: TVerboseLevel; Fmt: string;
              const Args: Array of const)
```

Visibility: public

Description: `Log` outputs formatted log messages with specified verbosity levels during build operations. This method provides structured logging with level-based filtering and message formatting capabilities.

The logging system supports various levels including debug, info, warning, and error messages, with optional format string processing for detailed diagnostic output.

29.13.50 TBuildEngine.Verbose

Synopsis: Controls verbose output during build operations

Declaration:
```
Property Verbose : Boolean
```

Visibility: public

Access: Read,Write

Description: When `True`, the build engine produces detailed output about compilation processes, file operations, and build steps.

When `False`, only essential messages and errors are displayed.

29.13.51 TBuildEngine.Interactive

Synopsis: Whether the build engine runs in interactive mode

Declaration: `Property Interactive : Boolean`

Visibility: public

Access: Read,Write

Description: `Interactive` controls whether the build engine prompts for user input during build operations. When `True`, the engine may ask for confirmations, choices, or input during the build process.

Interactive mode is useful during development and testing, while non-interactive mode is preferred for automated builds and continuous integration scenarios.

29.13.52 TBuildEngine.ListMode

Synopsis: Mode for listing package information

Declaration: `Property ListMode : Boolean`

Visibility: public

Access: Read,Write

Description: `ListMode` controls the format and detail level when listing package information and contents. This property determines how package listing output is formatted and what information is displayed.

Different list modes provide varying levels of detail from simple package names to information including dependencies, targets, and configuration details.

29.13.53 TBuildEngine.ForceCompile

Synopsis: Whether to force recompilation of all targets

Declaration: `Property ForceCompile : Boolean`

Visibility: public

Access: Read,Write

Description: `ForceCompile` enables complete recompilation of all targets regardless of file modification times. When `True`, all source files are recompiled even if outputs appear up-to-date.

Forced compilation ensures clean builds and is useful when dependency tracking may be unreliable or when build system changes require complete regeneration of all outputs.

29.13.54 TBuildEngine.ExternalPackages

Synopsis: Collection of external package dependencies

Declaration: `Property ExternalPackages : TPackages`

Visibility: public

Access: Read

Description: `ExternalPackages` contains the collection of external package dependencies required by the build process. This property manages packages that are not built by the current build system but are required for compilation.

External packages include system libraries, third-party components, and pre-compiled packages that must be available for successful compilation and linking.

29.13.55 TBuildEngine.StartDir

Synopsis: Starting directory for build operations

Declaration: `Property StartDir : string`

Visibility: public

Access: Read

Description: `StartDir` specifies the base directory from which all build operations are initiated. This property establishes the root path for resolving relative file paths and organizing build outputs.

The starting directory serves as the reference point for source file locations, output directories, and configuration file paths during the build process.

29.13.56 TBuildEngine.NotifyEventCollection

Synopsis: Collection of notification event handlers

Declaration: `Property NotifyEventCollection : TNotifyEventCollection`

Visibility: public

Access: Read

Description: `NotifyEventCollection` manages the collection of event handlers that receive notifications during build operations. This property enables registration of multiple event listeners for build progress monitoring.

Event handlers can monitor compilation progress, installation status, and other build events for logging, user interface updates, and custom processing requirements.

29.13.57 TBuildEngine.OnLog

Synopsis: Event handler for logging messages

Declaration: `Property OnLog : TLogEvent`

Visibility: public

Access: Read,Write

Description: `OnLog` is the event handler called when the build engine generates log messages. This event enables custom logging implementations and integration with external logging systems.

The event handler receives log level, message text, and formatting parameters, allowing custom processing, filtering, and routing of build system messages.

29.14 TCommand

29.14.1 Description

Class that encapsulates a command to be executed at specific points during the build process. Stores command information including the executable, arguments, and execution timing.

29.14.2 Method overview

Page	Method	Description
957	CmdLineOptions	Get formatted command line options
957	Destroy	Destructor that cleans up command resources
957	HaveOptions	Check if command has any options
958	ParseOptions	Parse options from a string

29.14.3 Property overview

Page	Properties	Access	Description
959	AfterCommand	rw	Command to execute after this command
959	At	rw	When to execute the command
959	BeforeCommand	rw	Command to execute before this command
958	Command	rw	Command line to execute
958	DestFile	rw	Destination file for the command
959	IgnoreResult	rw	Whether to ignore command execution result
958	Options	rw	Command line options
958	SourceFile	rw	Source file for the command

29.14.4 TCommand.Destroy

Synopsis: Destructor that cleans up command resources

Declaration: `destructor Destroy;  Override`

Visibility: public

Description: Destructor that releases memory and resources allocated by the command object. Ensures proper cleanup of internal data structures.

29.14.5 TCommand.HaveOptions

Synopsis: Check if command has any options

Declaration: `function HaveOptions : Boolean`

Visibility: public

Description: Function that returns `True` if the command has any command-line options or arguments configured. Used to determine if additional formatting is needed when building the command line.

29.14.6 TCommand.CmdLineOptions

Synopsis: Get formatted command line options

Declaration: `function CmdLineOptions : string`

Visibility: public

Description: Function that builds and returns a formatted string containing all command-line options and arguments. Used when constructing the final command line for execution.

29.14.7 TCommand.ParseOptions

Synopsis: Parse options from a string

Declaration: `procedure ParseOptions(S: string)`

Visibility: public

Description: Procedure that parses command-line options from a string and stores them internally. Breaks down the options string into individual arguments for later use in command execution.

29.14.8 TCommand.SourceFile

Synopsis: Source file for the command

Declaration: `Property SourceFile : string`

Visibility: public

Access: Read,Write

Description: Property specifying the source file that the command operates on. Used for commands that process or transform specific files during the build process.

29.14.9 TCommand.DestFile

Synopsis: Destination file for the command

Declaration: `Property DestFile : string`

Visibility: public

Access: Read,Write

Description: Property specifying the destination file that the command produces or writes to. Used for commands that generate output files during the build process.

29.14.10 TCommand.Command

Synopsis: Command line to execute

Declaration: `Property Command : string`

Visibility: public

Access: Read,Write

Description: Property containing the base command or executable name to be executed. Combined with options and arguments to form the complete command line.

29.14.11 TCommand.Options

Synopsis: Command line options

Declaration: `Property Options : TStrings`

Visibility: public

Access: Read,Write

Description: String list property containing command-line options and arguments. Each item represents a separate command-line parameter or option.

29.14.12 TCommand.At

Synopsis: When to execute the command

Declaration: `Property At : TCommandAt`

Visibility: public

Access: Read,Write

Description: Property of type TCommandAt (918) that specifies when during the build process this command should be executed. Determines the execution timing relative to compilation, installation, or other build phases.

29.14.13 TCommand.IgnoreResult

Synopsis: Whether to ignore command execution result

Declaration: `Property IgnoreResult : Boolean`

Visibility: public

Access: Read,Write

Description: Boolean property that determines whether command execution failures should be ignored. When `True`, the build process continues even if the command returns an error code.

29.14.14 TCommand.BeforeCommand

Synopsis: Command to execute before this command

Declaration: `Property BeforeCommand : TNotifyEvent`

Visibility: public

Access: Read,Write

Description: Property specifying another command that should be executed immediately before this command. Allows chaining of command execution for complex build operations.

29.14.15 TCommand.AfterCommand

Synopsis: Command to execute after this command

Declaration: `Property AfterCommand : TNotifyEvent`

Visibility: public

Access: Read,Write

Description: Property specifying another command that should be executed immediately after this command. Allows chaining of command execution for complex build operations.

29.15 TCommandEnumerator

29.15.1 Description

Enumerator class that enables for..in loop iteration over command collections. Provides standard iterator interface for traversing TCommand (956) objects.

29.15.2 Method overview

Page	Method	Description
960	GetCurrent	Get current command in enumeration

29.15.3 Property overview

Page	Properties	Access	Description
960	Current	r	Current command in enumeration

29.15.4 TCommandEnumerator.GetCurrent

Synopsis: Get current command in enumeration

Declaration: `function GetCurrent : TCommand`

Visibility: public

Description: Function that returns the current TCommand (956) object at the enumerator position. Called by the for..in loop mechanism to access the current item.

29.15.5 TCommandEnumerator.Current

Synopsis: Current command in enumeration

Declaration: `Property Current : TCommand`

Visibility: public

Access: Read

Description: `Current` returns the current TCommand (956) object in the enumeration sequence.

This property is used in for-in loops to access each command as the enumerator iterates through the collection.

29.16 TCommands

29.16.1 Description

`TCommands` is a specialized collection class that manages a list of TCommand (956) objects.

This class provides methods to add, remove, and iterate through build commands that should be executed during the package compilation process.

Commands are typically shell commands, compiler invocations, or other external tools needed for building packages.

29.16.2 Method overview

Page	Method	Description
961	AddCommand	Add a new command to the collection
961	GetEnumerator	Get enumerator for iterating commands

29.16.3 Property overview

Page	Properties	Access	Description
961	CommandItems	rw	Indexed access to command items
962	Commands	r	Access to commands by destination
962	DefaultAt	rw	Default execution timing for commands

29.16.4 TCommands.AddCommand

Synopsis: Add a new command to the collection

Declaration:
```
function AddCommand(const Cmd: string) : TCommand
function AddCommand(const Cmd: string; const Options: string) : TCommand
function AddCommand(const Cmd: string; const Options: string;
                    const Dest: string; const Source: string) : TCommand
function AddCommand(At: TCommandAt; const Cmd: string) : TCommand
function AddCommand(At: TCommandAt; const Cmd: string;
                    const Options: string) : TCommand
function AddCommand(At: TCommandAt; const Cmd: string;
                    const Options: string; const Dest: string;
                    const Source: string) : TCommand
```

Visibility: public

Description: Creates and adds a new `TCommand` object to the commands collection. The command can be configured with executable path, options, source and destination files, and execution timing.

Commands are executed during the build process at the specified timing (before or after compilation, installation, etc.).

29.16.5 TCommands.GetEnumerator

Synopsis: Get enumerator for iterating commands

Declaration:
```
function GetEnumerator : TCommandEnumerator
```

Visibility: public

Description: Function that returns a TCommandEnumerator (959) object for iterating through the commands collection. Enables the use of for..in loops to iterate over all commands in the collection.

29.16.6 TCommands.CommandItems

Synopsis: Indexed access to command items

Declaration:
```
Property CommandItems[Index: Integer]: TCommand
```

Visibility: public

Access: Read,Write

Description: Indexed property providing direct access to TCommand (956) objects by their zero-based index. Allows reading and writing of command objects at specific positions in the collection.

29.16.7 TCommands.Commands

Synopsis: Access to commands by destination

Declaration:
```
Property Commands[Dest: string]: TCommand; default
```

Visibility: public

Access: Read

Description: Property providing access to commands based on their destination file or path. Allows retrieving commands that operate on specific destination targets.

29.16.8 TCommands.DefaultAt

Synopsis: Default execution timing for commands

Declaration:
```
Property DefaultAt : TCommandAt
```

Visibility: public

Access: Read,Write

Description: Property specifying the default TCommandAt (918) timing for newly created commands. Used when no explicit execution timing is specified for new commands.

29.17 TCompileWorkerThread

29.17.1 Description

`TCompileWorkerThread` implements a worker thread for executing compilation operations in parallel. This class enables concurrent compilation of multiple targets to improve build performance on multi-core systems.

The worker thread manages compilation tasks, handles thread synchronization, and reports progress back to the main thread for coordinated parallel build processing.

29.17.2 Method overview

Page	Method	Description
962	Create	Creates compilation worker thread
963	Destroy	Destroys worker thread and cleans up resources

29.17.3 TCompileWorkerThread.Create

Synopsis: Creates compilation worker thread

Declaration:
```
constructor Create(ABuildEngine: TBuildEngine;
                   NotifyMainThreadEvent: PRTLEvent);  Virtual
```

Visibility: public

Description: `Create` initializes a new compilation worker thread with the specified build engine and notification callback. This constructor prepares the thread for parallel compilation operations.

The worker thread is configured to communicate with the main thread through the provided notification event.

29.17.4 TCompileWorkerThread.Destroy

Synopsis: Destroys worker thread and cleans up resources

Declaration: `destructor Destroy;  Override`

Visibility: public

Description: `Destroy` terminates the worker thread and frees all allocated resources including build engine references and thread synchronization objects. This destructor ensures proper cleanup of the compilation thread.

The destructor waits for pending operations to complete before freeing resources.

29.18 TConditionalDestString

29.18.1 Description

`TConditionalDestString` extends conditional strings to include destination path information.

Used for file operations that require both source values and target destinations based on CPU and OS conditions.

29.18.2 Property overview

Page	Properties	Access	Description
963	DestPath	rw	Destination path for the conditional string

29.18.3 TConditionalDestString.DestPath

Synopsis: Destination path for the conditional string

Declaration: `Property DestPath : string`

Visibility: public

Access: Read,Write

Description: Destination file or directory path where the conditional string value should be placed.

This path is used during installation and file copy operations.

29.19 TConditionalDestStrings

29.19.1 Description

`TConditionalDestStrings` is a specialized collection that manages multiple TConditionalDestString (963) objects.

This class provides methods for adding, searching, and retrieving conditional destination strings based on platform criteria.

It enables efficient management of platform-specific file installation paths and destinations in build systems.

29.19.2 Method overview

Page	Method	Description
964	Add	Adds conditional destination string with path mapping
964	Create	Creates conditional destination strings collection

29.19.3 Property overview

Page	Properties	Access	Description
964	ConditionalStrings	rw	Indexed access to conditional destination strings

29.19.4 TConditionalDestStrings.Create

Synopsis: Creates conditional destination strings collection

Declaration:
```
constructor Create(AClass: TConditionalDestStringClass)
```

Visibility: public

Description: `Create` initializes a new TConditionalDestStrings (963) collection instance.

The constructor sets up an empty collection ready to accept conditional destination string items.

The collection inherits from the base collection classes and provides specialized functionality for destination string management.

29.19.5 TConditionalDestStrings.Add

Synopsis: Adds conditional destination string with path mapping

Declaration:
```
function Add(const Value: string; ADestPath: string)
                : TConditionalDestString
function Add(const Value: string; const OSes: TOSes; ADestPath: string)
                : TConditionalDestString
function Add(const Value: string; const CPUs: TCPUS; const OSes: TOSes;
              ADestPath: string) : TConditionalDestString
```

Visibility: public

Description: `Add` creates and adds a new conditional destination string with the specified value, destination path, and platform restrictions.

The string will only be used when the current build targets match the specified CPU architectures and operating systems.

This method enables platform-specific file installation paths and destination mapping in build configurations.

29.19.6 TConditionalDestStrings.ConditionalStrings

Synopsis: Indexed access to conditional destination strings

Declaration:
```
Property ConditionalStrings[Index: Integer]: TConditionalDestString; default
```

Visibility: public

Access: Read,Write

Description: `ConditionalStrings` provides indexed access to individual TConditionalDestString (963) items in the collection.

This property enables array-style access to retrieve or modify specific conditional destination strings by their position.

The index must be within the valid range from 0 to Count-1 to avoid range check errors.

29.20 TConditionalString

29.20.1 Description

`TConditionalString` represents a string value that can be conditionally applied based on target CPU architecture and operating system. This enables platform-specific configuration in build scripts.

The class provides matching functionality to determine if the string applies to a specific CPU/OS combination, making it the foundation for cross-platform build support.

See also: TConditionalStrings (966), TCpu (920), TOS (926)

29.20.2 Method overview

Page	Method	Description
965	Create	

29.20.3 Property overview

Page	Properties	Access	Description
966	CPUs	rw	CPU architectures for which this string applies
966	OSes	rw	Operating systems for which this string applies
965	Value	rw	String value for this condition

29.20.4 TConditionalString.Create

Declaration: `constructor Create;  Virtual`

Visibility: public

29.20.5 TConditionalString.Value

Synopsis: String value for this condition

Declaration: `Property Value : string`

Visibility: public

Access: Read,Write

Description: `Value` contains the actual string content that is used when the conditional criteria are met.

This property stores the string that will be returned or applied when the CPU and OS conditions match the target platform.

The value can contain any string content including file paths, compiler options, or configuration parameters.

29.20.6 TConditionalString.OSes

Synopsis: Operating systems for which this string applies

Declaration: `Property OSes : TOSes`

Visibility: public

Access: Read,Write

Description: `OSes` specifies the set of operating systems for which this conditional string is applicable.

This property contains a list of TOS (926) values that define the target operating systems.

If empty, the string applies to all operating systems, otherwise it only applies to the specified OS platforms.

29.20.7 TConditionalString.CPUs

Synopsis: CPU architectures for which this string applies

Declaration: `Property CPUs : TCPUS`

Visibility: public

Access: Read,Write

Description: `CPUs` specifies the set of CPU architectures for which this conditional string is applicable.

This property contains a list of TCpu (920) values that define the target processor architectures.

If empty, the string applies to all CPU architectures, otherwise it only applies to the specified processor types.

29.21 TConditionalStrings

29.21.1 Description

`TConditionalStrings` is a specialized collection that manages multiple TConditionalString (965) objects.

This class provides methods for adding, searching, and retrieving conditional strings based on platform criteria.

It enables efficient management of platform-specific configuration values in build systems.

29.21.2 Method overview

Page	Method	Description
967	Add	Add a new conditional string
967	Create	

29.21.3 Property overview

Page	Properties	Access	Description
967	ConditionalStrings	rw	Indexed access to conditional string items

29.21.4 TConditionalStrings.Create

Declaration: `constructor Create(AClass: TConditionalStringClass)`

Visibility: public

29.21.5 TConditionalStrings.Add

Synopsis: Add a new conditional string

Declaration:
```
function Add(const Value: string) : TConditionalString
function Add(const Value: string; const OSes: TOSes)
             : TConditionalString
function Add(const Value: string; const CPUs: TCPUS; const OSes: TOSes)
             : TConditionalString
```

Visibility: public

Description: Creates and adds a new conditional string with the specified value and platform restrictions. The string will only be used when the current build targets match the specified CPU architectures and operating systems.

Platform-specific strings enable configuration of different values for different target platforms within the same package.

29.21.6 TConditionalStrings.ConditionalStrings

Synopsis: Indexed access to conditional string items

Declaration: `Property ConditionalStrings[Index: Integer]: TConditionalString; default`

Visibility: public

Access: Read,Write

Description: Provides indexed access to individual conditional string items in the collection.

Each item contains a value and associated CPU/OS conditions.

29.22 TCustomDefaults

29.22.1 Description

`TCustomDefaults` serves as the base class for managing default configuration settings in the fpmake build system. This abstract class provides a foundation for storing and accessing system-wide defaults and build preferences.

Derived classes extend this base to provide specific configuration management for different aspects of the build system such as compiler settings, paths, and installation options.

29.22.2 Method overview

Page	Method	Description
971	CompilerDefaults	Sets up compiler default configuration
969	Create	Creates a new defaults configuration object
970	Destroy	Destroys the defaults object and releases resources
970	HaveOptions	Checks if configuration options are available
970	InitDefaults	Initializes default configuration values
970	IsBuildDifferentFromTarget	Checks if build configuration differs from target
971	LoadFromFile	Loads configuration settings from a file
972	LoadFromStream	Loads configuration settings from a stream
971	LocalInit	Initializes configuration from local file
971	SaveToFile	Saves configuration settings to a file
971	SaveToStream	Saves configuration settings to a stream

29.22.3 Property overview

Page	Properties	Access	Description
981	Archive	rw	Archive creation command and options
976	BaseInstallDir	rw	Base directory for installations
977	BinInstallDir	rw	Directory for installing binary files
974	BuildCPU	r	CPU architecture for the build environment
981	BuildMode	rw	Build mode configuration setting
973	BuildOS	r	Operating system for the build environment
973	BuildString	r	Build configuration string
979	Compiler	rw	Compiler executable path and options
973	CompilerVersion	r	Version of the compiler being used
979	Copy	rw	File copy command and options
972	CPU	rw	Target CPU architecture
978	DocInstallDir	rw	Directory for installing documentation files
978	ExamplesInstallDir	rw	Directory for installing example files
973	ExplicitOSNone	rw	Whether to explicitly specify no target OS
978	FPDocOptions	rw	Command line options for FPDoc
978	FPDocOutputDir	rw	Output directory for FPDoc documentation
979	FPUnitSourcePath	rw	Source path for FPUnit testing framework
975	GlobalUnitDir	rw	Global directory for compiled units
981	IgnoreInvalidOptions	rw	Whether to ignore invalid configuration options
982	InstallExamples	rw	Whether to install example files
977	LibInstallDir	rw	Directory for installing library files
975	LocalUnitDir	rw	Local directory for compiled units
980	MkDir	rw	Directory creation command and options
974	Mode	rw	Build mode configuration
979	Move	rw	File move command and options
975	NoFPCCfg	rw	Whether to ignore FPC configuration files
974	Options	rw	Configuration options for the build
972	OS	rw	Target operating system
976	Prefix	rw	Installation prefix directory
980	Remove	rw	File removal command and options
980	RemoveDir	rw	Directory removal command and options
980	RemoveTree	rw	Directory tree removal command and options
976	SearchPath	rw	Search paths for finding files
982	SingleFPDocFile	rw	Whether to generate single FPDoc output file
982	SkipAllPrograms	rw	Whether to skip building all programs
982	SkipCrossPrograms	rw	Whether to skip cross-compilation programs
972	Target	rw	Build target specification
975	ThreadsAmount	rw	Number of threads to use during compilation
977	UnitConfigFilesInstallDir	rw	Directory for installing unit configuration files
977	UnitInstallDir	rw	Directory for installing unit files
974	UnixPaths	rw	Whether to use Unix-style path conventions
981	UseEnvironment	rw	Whether to use environment variables
976	ZipPrefix	rw	Prefix for archive file operations

29.22.4 TCustomDefaults.Create

Synopsis: Creates a new defaults configuration object

Declaration: `constructor Create`

Visibility: public

Description: `Create` initializes a new `TCustomDefaults` instance with default configuration values. This constructor sets up the basic configuration structure for the fpmake build system.

The created object serves as a container for various build system defaults including paths, compiler options, and installation preferences.

29.22.5 TCustomDefaults.Destroy

Synopsis: Destroys the defaults object and releases resources

Declaration: `destructor Destroy;  Override`

Visibility: public

Description: Destroys the `TCustomDefaults` instance and releases all associated resources including configuration collections and internal data structures.

29.22.6 TCustomDefaults.InitDefaults

Synopsis: Initializes default configuration values

Declaration: `procedure InitDefaults`

Visibility: public

Description: Sets up the default configuration values for this defaults object. This method is called during initialization to establish baseline configuration settings that can be overridden by specific values.

29.22.7 TCustomDefaults.HaveOptions

Synopsis: Checks if configuration options are available

Declaration: `function HaveOptions : Boolean`

Visibility: public

Description: Returns `True` if this defaults object contains configuration options that can be applied. This is used to determine whether additional configuration processing is needed.

29.22.8 TCustomDefaults.IsBuildDifferentFromTarget

Synopsis: Checks if build configuration differs from target

Declaration: `function IsBuildDifferentFromTarget : Boolean`

Visibility: public

Description: Returns `True` if the current build configuration differs from the target's default configuration. This is used to determine whether custom build settings need to be applied.

29.22.9 TCustomDefaults.CompilerDefaults

Synopsis: Sets up compiler default configuration

Declaration: `procedure CompilerDefaults;  Virtual`

Visibility: public

Description: Initializes the default compiler configuration settings including standard options, search paths, and compilation flags that are applied by default to all builds using these defaults.

29.22.10 TCustomDefaults.LocalInit

Synopsis: Initializes configuration from local file

Declaration: `procedure LocalInit(const AFileName: string)`

Visibility: public

Description: Loads configuration settings from a local initialization file. This allows customization of default settings based on local environment or user preferences.

29.22.11 TCustomDefaults.LoadFromFile

Synopsis: Loads configuration settings from a file

Declaration: `procedure LoadFromFile(const AFileName: string)`

Visibility: public

Description: Loads configuration settings from the specified file, replacing the current default configuration. The file should contain configuration options in a format compatible with the defaults system.

29.22.12 TCustomDefaults.SaveToFile

Synopsis: Saves configuration settings to a file

Declaration: `procedure SaveToFile(const AFileName: string)`

Visibility: public

Description: Saves the current default configuration settings to the specified file. This allows preservation of customized default settings for future use or distribution.

29.22.13 TCustomDefaults.SaveToStream

Synopsis: Saves configuration settings to a stream

Declaration: `procedure SaveToStream(S: TStream);  Virtual`

Visibility: public

Description: Saves the current default configuration settings to the provided stream. This allows flexible output of configuration data to various stream-based destinations.

29.22.14 TCustomDefaults.LoadFromStream

Synopsis: Loads configuration settings from a stream

Declaration: `procedure LoadFromStream(S: TStream);  Virtual`

Visibility: public

Description: Loads default configuration settings from the provided stream. This enables flexible input of configuration data from various stream-based sources.

The loaded settings become the new defaults for subsequent build operations.

29.22.15 TCustomDefaults.Target

Synopsis: Build target specification

Declaration: `Property Target : string`

Visibility: public

Access: Read,Write

Description: Specifies the build target name or identifier for default configuration.

This property is used to identify which target-specific settings should be applied during the build process.

29.22.16 TCustomDefaults.OS

Synopsis: Target operating system

Declaration: `Property OS : TOS`

Visibility: public

Access: Read,Write

Description: Specifies the target operating system for compilation (Windows, Linux, Darwin, etc.).

This property determines OS-specific compiler settings and conditional compilation directives that are applied during the build process.

29.22.17 TCustomDefaults.CPU

Synopsis: Target CPU architecture

Declaration: `Property CPU : TCpu`

Visibility: public

Access: Read,Write

Description: Specifies the target CPU architecture for compilation (i386, x86_64, ARM, etc.).

This property determines CPU-specific compiler settings and optimizations that are applied during the build process.

29.22.18 TCustomDefaults.CompilerVersion

Synopsis: Version of the compiler being used

Declaration: `Property CompilerVersion : string`

Visibility: public

Access: Read

Description: Contains the version string of the Free Pascal compiler being used for builds.

This property provides version information for compatibility checking and build environment reporting.

29.22.19 TCustomDefaults.ExplicitOSNone

Synopsis: Whether to explicitly specify no target OS

Declaration: `Property ExplicitOSNone : Boolean`

Visibility: public

Access: Read,Write

Description: Boolean flag that controls whether to explicitly specify no target operating system.

When `True`, the build system will not set any OS-specific compilation flags, resulting in generic or OS-independent compilation.

29.22.20 TCustomDefaults.BuildString

Synopsis: Build configuration string

Declaration: `Property BuildString : string`

Visibility: public

Access: Read

Description: String representation of the current build configuration settings.

This property provides a human-readable summary of the target platform, compiler options, and build parameters.

29.22.21 TCustomDefaults.BuildOS

Synopsis: Operating system for the build environment

Declaration: `Property BuildOS : TOS`

Visibility: public

Access: Read

Description: Specifies the operating system of the build environment (the system performing the compilation).

This property is used for cross-compilation scenarios where the build OS differs from the target OS.

29.22.22 TCustomDefaults.BuildCPU

Synopsis: CPU architecture for the build environment

Declaration: `Property BuildCPU : TCpu`

Visibility: public

Access: Read

Description: Specifies the CPU architecture of the build environment (the system performing the compilation).

This property is used for cross-compilation scenarios where the build CPU differs from the target CPU architecture.

29.22.23 TCustomDefaults.Mode

Synopsis: Build mode configuration

Declaration: `Property Mode : TCompilerMode`

Visibility: public

Access: Read,Write

Description: Specifies the build mode that controls optimization, debugging, and code generation settings.

Common modes include debug (with symbols and checks) and release (optimized for performance).

29.22.24 TCustomDefaults.UnixPaths

Synopsis: Whether to use Unix-style path conventions

Declaration: `Property UnixPaths : Boolean`

Visibility: public

Access: Read,Write

Description: Boolean flag that controls whether to use Unix-style path separators and conventions.

When `True`, forward slashes are used for paths instead of platform-specific separators, enabling cross-platform path consistency.

29.22.25 TCustomDefaults.Options

Synopsis: Configuration options for the build

Declaration: `Property Options : TStrings`

Visibility: public

Access: Read,Write

Description: Collection of configuration options that are applied to all builds using these defaults.

Options include compiler flags, preprocessor definitions, optimization settings, and other build-specific parameters.

29.22.26 TCustomDefaults.NoFPCCfg

Synopsis: Whether to ignore FPC configuration files

Declaration: `Property NoFPCCfg : Boolean`

Visibility: public

Access: Read,Write

Description: Boolean flag that controls whether to ignore system-wide Free Pascal configuration files.

When `True`, the build system bypasses automatic loading of fpc.cfg and similar configuration files.

29.22.27 TCustomDefaults.ThreadsAmount

Synopsis: Number of threads to use during compilation

Declaration: `Property ThreadsAmount : Integer`

Visibility: public

Access: Read,Write

Description: Specifies the number of parallel threads to use during multi-threaded compilation.

Higher thread counts can improve build performance on multi-core systems, while lower counts reduce resource usage.

29.22.28 TCustomDefaults.LocalUnitDir

Synopsis: Local directory for compiled units

Declaration: `Property LocalUnitDir : string`

Visibility: public

Access: Read,Write

Description: Specifies the local directory where compiled unit files (.ppu, .o) are stored.

This directory is used for unit output during compilation and serves as a search path for unit dependencies in the same package.

29.22.29 TCustomDefaults.GlobalUnitDir

Synopsis: Global directory for compiled units

Declaration: `Property GlobalUnitDir : string`

Visibility: public

Access: Read,Write

Description: Specifies the global directory where system-wide compiled unit files are located.

This directory is searched for unit dependencies that are not part of the current package, such as RTL and FCL units.

29.22.30 TCustomDefaults.SearchPath

Synopsis: Search paths for finding files

Declaration: `Property SearchPath : TStrings`

Visibility: public

Access: Read,Write

Description: Specifies the search paths used for locating source files, units, and other resources.

Multiple paths can be specified and are searched in order when resolving file dependencies during compilation.

29.22.31 TCustomDefaults.Prefix

Synopsis: Installation prefix directory

Declaration: `Property Prefix : string`

Visibility: public

Access: Read,Write

Description: Specifies the base installation prefix directory for all package installations.

All installation paths are relative to this prefix, enabling flexible deployment to different base locations.

29.22.32 TCustomDefaults.ZipPrefix

Synopsis: Prefix for archive file operations

Declaration: `Property ZipPrefix : string`

Visibility: public

Access: Read,Write

Description: Specifies the directory prefix used when creating or extracting archive files.

This prefix is applied to file paths within archives to ensure proper relative path structure during archive operations.

29.22.33 TCustomDefaults.BaseInstallDir

Synopsis: Base directory for installations

Declaration: `Property BaseInstallDir : string`

Visibility: public

Access: Read,Write

Description: Specifies the base directory where all package components are installed.

This directory serves as the root for all installation operations and contains subdirectories for units, documentation, and examples.

29.22.34 TCustomDefaults.UnitInstallDir

Synopsis: Directory for installing unit files

Declaration: `Property UnitInstallDir : string`

Visibility: public

Access: Read,Write

Description: Specifies the target directory where compiled unit files are installed.

This directory contains the final .ppu and .o files that are installed for use by other packages and applications.

29.22.35 TCustomDefaults.UnitConfigFilesInstallDir

Synopsis: Directory for installing unit configuration files

Declaration: `Property UnitConfigFilesInstallDir : string`

Visibility: public

Access: Read,Write

Description: Specifies the target directory where unit configuration files are installed.

These configuration files contain search paths and settings that help the compiler locate and use the installed units.

29.22.36 TCustomDefaults.BinInstallDir

Synopsis: Directory for installing binary files

Declaration: `Property BinInstallDir : string`

Visibility: public

Access: Read,Write

Description: Specifies the target directory where executable binary files are installed.

This directory contains compiled programs, utilities, and tools that are part of the package distribution.

29.22.37 TCustomDefaults.LibInstallDir

Synopsis: Directory for installing library files

Declaration: `Property LibInstallDir : string`

Visibility: public

Access: Read,Write

Description: Specifies the target directory where library files are installed.

This directory contains shared libraries, static libraries, and other linkable library components provided by the package.

29.22.38 TCustomDefaults.DocInstallDir

Synopsis: Directory for installing documentation files

Declaration: `Property DocInstallDir : string`

Visibility: public

Access: Read,Write

Description: Specifies the target directory where documentation files are installed.

This directory contains user manuals, API documentation, readme files, and other informational content for the package.

29.22.39 TCustomDefaults.ExamplesInstallDir

Synopsis: Directory for installing example files

Declaration: `Property ExamplesInstallDir : string`

Visibility: public

Access: Read,Write

Description: Specifies the target directory where example files are installed.

This directory contains sample programs, demonstration code, and tutorials that show how to use the package functionality.

29.22.40 TCustomDefaults.FPDocOutputDir

Synopsis: Output directory for FPDoc documentation

Declaration: `Property FPDocOutputDir : string`

Visibility: public

Access: Read,Write

Description: Specifies the output directory where FPDoc-generated documentation is created.

This directory receives the HTML, XML, or other formatted documentation files generated by the FPDoc documentation system.

29.22.41 TCustomDefaults.FPDocOptions

Synopsis: Command line options for FPDoc

Declaration: `Property FPDocOptions : string`

Visibility: public

Access: Read,Write

Description: Specifies command line options and parameters passed to the FPDoc documentation generator.

These options control documentation format, output settings, processing modes, and other FPDoc-specific configuration parameters.

29.22.42 TCustomDefaults.FPUnitSourcePath

Synopsis: Source path for FPUnit testing framework

Declaration: `Property FPUnitSourcePath : string`

Visibility: public

Access: Read,Write

Description: Specifies the path to the FPUnit testing framework source files.

This path is used to locate FPUnit units and resources when building test programs and test suites for package validation.

29.22.43 TCustomDefaults.Compiler

Synopsis: Compiler executable path and options

Declaration: `Property Compiler : string`

Visibility: public

Access: Read,Write

Description: Specifies the Free Pascal compiler executable path and default compilation options.

This property configures the compiler command, version, and global settings used for all compilation operations in the build system.

29.22.44 TCustomDefaults.Copy

Synopsis: File copy command and options

Declaration: `Property Copy : string`

Visibility: public

Access: Read,Write

Description: Specifies the system command and options used for file copy operations.

This property configures the file copying mechanism used during installation, archive creation, and other file management tasks.

29.22.45 TCustomDefaults.Move

Synopsis: File move command and options

Declaration: `Property Move : string`

Visibility: public

Access: Read,Write

Description: Specifies the system command and options used for file move operations.

This property configures the file moving mechanism used for reorganizing files during build and installation processes.

29.22.46 TCustomDefaults.Remove

Synopsis: File removal command and options

Declaration: `Property Remove : string`

Visibility: public

Access: Read,Write

Description: Specifies the system command and options used for file removal operations.

This property configures the file deletion mechanism used during cleanup, uninstallation, and other file management tasks.

29.22.47 TCustomDefaults.RemoveDir

Synopsis: Directory removal command and options

Declaration: `Property RemoveDir : string`

Visibility: public

Access: Read,Write

Description: Specifies the system command and options used for empty directory removal.

This property configures the directory deletion mechanism used when removing empty directories during cleanup operations.

29.22.48 TCustomDefaults.RemoveTree

Synopsis: Directory tree removal command and options

Declaration: `Property RemoveTree : string`

Visibility: public

Access: Read,Write

Description: Specifies the system command and options used for recursive directory tree removal.

This property configures the mechanism for deleting entire directory structures with all contained files and subdirectories.

29.22.49 TCustomDefaults.MkDir

Synopsis: Directory creation command and options

Declaration: `Property MkDir : string`

Visibility: public

Access: Read,Write

Description: Specifies the system command and options used for directory creation operations.

This property configures the directory creation mechanism used during installation and build processes to ensure required directories exist.

29.22.50 TCustomDefaults.Archive

Synopsis: Archive creation command and options

Declaration: `Property Archive : string`

Visibility: public

Access: Read,Write

Description: Specifies the system command and options used for archive file creation.

This property configures the archiving mechanism used to create TAR, ZIP, or other compressed archive files for package distribution.

29.22.51 TCustomDefaults.UseEnvironment

Synopsis: Whether to use environment variables

Declaration: `Property UseEnvironment : Boolean`

Visibility: public

Access: Read,Write

Description: Boolean flag that controls whether to use environment variables in configuration.

When `True`, environment variables are expanded in paths and options, enabling dynamic configuration based on the system environment.

29.22.52 TCustomDefaults.IgnoreInvalidOptions

Synopsis: Whether to ignore invalid configuration options

Declaration: `Property IgnoreInvalidOptions : Boolean`

Visibility: public

Access: Read,Write

Description: Boolean flag that controls whether invalid or unrecognized options are ignored.

When `True`, unknown configuration options do not cause build failures but are silently ignored with optional warning messages.

29.22.53 TCustomDefaults.BuildMode

Synopsis: Build mode configuration setting

Declaration: `Property BuildMode : TBuildMode`

Visibility: public

Access: Read,Write

Description: `BuildMode` specifies the current build mode configuration for the package compilation process. This setting influences compiler optimization levels, debug information inclusion, and other build parameters.

Common build modes include debug, release, and custom configurations tailored for specific build requirements.

29.22.54 TCustomDefaults.SingleFPDocFile

Synopsis: Whether to generate single FPDoc output file

Declaration: `Property SingleFPDocFile : Boolean`

Visibility: public

Access: Read,Write

Description: `SingleFPDocFile` controls whether fpdoc generates documentation as a single combined file or as separate files for each unit. When `True`, all documentation is merged into one output file.

Single file output is useful for creating documentation packages or when distributing documentation as a single downloadable file.

29.22.55 TCustomDefaults.InstallExamples

Synopsis: Whether to install example files

Declaration: `Property InstallExamples : Boolean`

Visibility: public

Access: Read,Write

Description: `InstallExamples` controls whether example files are installed along with the package. When `True`, example source code and demonstration files are copied to the installation directory.

Installing examples provides users with ready-to-use code samples that demonstrate package functionality and serve as a learning resource.

29.22.56 TCustomDefaults.SkipCrossPrograms

Synopsis: Whether to skip cross-compilation programs

Declaration: `Property SkipCrossPrograms : Boolean`

Visibility: public

Access: Read,Write

Description: `SkipCrossPrograms` determines whether program targets are skipped during cross-compilation builds. When `True`, only libraries and units are compiled when building for different target architectures.

This setting is useful when the target platform cannot execute the compiled programs or when only library components are needed for cross-platform deployment.

29.22.57 TCustomDefaults.SkipAllPrograms

Synopsis: Whether to skip building all programs

Declaration: `Property SkipAllPrograms : Boolean`

Visibility: public

Access: Read,Write

Description: `SkipAllPrograms` controls whether all program targets are excluded from the build process. When `True`, only library units are compiled, skipping executable programs entirely.

This option is useful for library-only distributions or when building packages that provide only reusable components without standalone executables.

29.23 TCustomInstaller

29.23.1 Description

`TCustomInstaller` is the base class for implementing package installation systems. It provides the framework for managing packages, coordinating with the build engine, and handling different installation modes.

The installer manages a collection of packages, provides access to the build engine, and supports different run modes including compilation, installation, cleaning, and archiving. It serves as the bridge between the command-line interface and the build system.

See also: TBuildEngine (941), TPackage (1009), TPackages (1035), TRunMode (927)

29.23.2 Method overview

Page	Method	Description
984	AddPackage	Adds package to installer collection
984	AddPackageVariant	Adds package variant to installer
983	Create	Creates a new installer instance
984	destroy	Destroys installer instance and frees resources
984	Run	Executes the installer with the current configuration

29.23.3 Property overview

Page	Properties	Access	Description
985	BuildEngine	r	Build engine instance used for compilation and installation
985	FPMakeOptionsString	r	String representation of FPMake options
986	ListMode	r	Mode for listing package information
986	NotifyEventCollection	r	Collection of notification event handlers
985	Packages	r	Collection of packages managed by this installer
985	RunMode	r	Current operation mode of the installer

29.23.4 TCustomInstaller.Create

Synopsis: Creates a new installer instance

Declaration: `constructor Create(AOwner: TComponent); Virtual`

Visibility: public

Description: `Create` initializes a new `TCustomInstaller` instance with default configuration. This constructor sets up the installer framework including the build engine, package collection, and default settings.

It also analyzes the command-line to determine what the RunMode (985) is.

The created instance is then ready to have packages added and can be configured for different run modes such as compilation, installation, cleaning, or archiving that will be executed during the Run (984) procedure.

See also: RunMode (985), Run (984)

29.23.5 TCustomInstaller.destroy

Synopsis: Destroys installer instance and frees resources

Declaration: `destructor destroy;  Override`

Visibility: public

Description: `destroy` frees all resources allocated by the installer including the build engine and package collection. This destructor ensures proper cleanup of all installer components and allocated memory.

The destructor handles cleanup of temporary files, releases system resources, and properly terminates any ongoing build operations before freeing the installer instance.

29.23.6 TCustomInstaller.AddPackage

Synopsis: Adds package to installer collection

Declaration: `function AddPackage(const AName: string) : TPackage`

Visibility: public

Description: `AddPackage` creates and adds a new TPackage (1009) instance to the installer's package collection. The method returns the newly created package which can then be configured with targets, dependencies, and settings.

Each package added to the installer will be processed during the run operation according to the current run mode, whether for compilation, installation, or other operations.

29.23.7 TCustomInstaller.AddPackageVariant

Synopsis: Adds package variant to installer

Declaration:
```
function AddPackageVariant(AName: string; AIsInheritable: Boolean;
                           AutoAddToPackage: Boolean) : TPackageVariants
```

Visibility: public

Description: `AddPackageVariant` creates and adds a package variant that provides alternative build configurations. Package variants enable different compilation options, target platforms, or feature sets within the same package.

Variants can be inheritable, allowing derived packages to inherit variant settings, and can be automatically added to packages for convenient configuration management.

29.23.8 TCustomInstaller.Run

Synopsis: Executes the installer with the current configuration

Declaration: `function Run : Boolean`

Visibility: public

Description: Runs the installer according to the current RunMode (985) setting. This method coordinates the entire build process, from compilation through installation, depending on the specified run mode.

See also: RunMode (985), TBuildEngine (941)

29.23.9 TCustomInstaller.FPMakeOptionsString

Synopsis: String representation of FPMake options

Declaration: `Property FPMakeOptionsString : string`

Visibility: public

Access: Read

Description: `FPMakeOptionsString` provides a string representation of the current FPMake configuration options. This property formats the installer's settings into a readable string for logging and diagnostic purposes.

The string includes information about run mode, package configuration, build options, and other settings that affect the installer's operation.

29.23.10 TCustomInstaller.BuildEngine

Synopsis: Build engine instance used for compilation and installation

Declaration: `Property BuildEngine : TBuildEngine`

Visibility: public

Access: Read

Description: The `BuildEngine` property provides access to the TBuildEngine (941) instance that handles the actual compilation, dependency resolution, and file operations.

See also: TBuildEngine (941)

29.23.11 TCustomInstaller.Packages

Synopsis: Collection of packages managed by this installer

Declaration: `Property Packages : TPackages`

Visibility: public

Access: Read

Description: The `Packages` property contains the collection of TPackage (1009) instances that will be processed by this installer.

See also: TPackages (1035), TPackage (1009)

29.23.12 TCustomInstaller.RunMode

Synopsis: Current operation mode of the installer

Declaration: `Property RunMode : TRunMode`

Visibility: public

Access: Read

Description: `RunMode` determines the type of operation the installer will perform when `Run` is called. Possible modes include compilation, installation, cleaning, archiving, and other package management operations.

The run mode controls which build engine methods are invoked and how packages are processed during the installer execution cycle.

29.23.13 TCustomInstaller.ListMode

Synopsis: Mode for listing package information

Declaration: `Property ListMode : Boolean`

Visibility: public

Access: Read

Description: When `ListMode` is `True` then the installer will simply list the various commands it would normally execute to execute the action.

It is similar to `-dry-run` found in other utilities.

29.23.14 TCustomInstaller.NotifyEventCollection

Synopsis: Collection of notification event handlers

Declaration: `Property NotifyEventCollection : TNotifyEventCollection`

Visibility: public

Access: Read

Description: `NotifyEventCollection` manages the collection of event handlers that receive notifications during installer operations. This property enables registration of multiple event listeners for progress monitoring.

Event handlers can monitor build progress, installation status, and other installer events for logging, user interface updates, and custom processing requirements.

29.24 TDependencies

29.24.1 Description

`TDependencies` is a collection class that manages build dependencies for Free Pascal packages.

This collection stores and organizes dependency objects, providing indexed access and management functionality.

29.24.2 Method overview

Page	Method	Description
987	Add	Adds a new package dependency to the collection
988	AddInclude	Adds an include file dependency to the collection
987	AddUnit	Adds a unit dependency to the collection
987	GetDependency	Gets dependency at specified index
987	SetDependency	Sets dependency at specified index

29.24.3 Property overview

Page	Properties	Access	Description
988	Dependencies	rw	Indexed access to dependency objects in the collection

29.24.4 TDependencies.GetDependency

Synopsis: Gets dependency at specified index

Declaration:
```
function GetDependency(Index: Integer) : TDependency
```

Visibility: default

Description: Retrieves the TDependency (988) object at the specified index position within the dependencies collection.

Returns the dependency object for accessing its properties and configuration settings.

29.24.5 TDependencies.SetDependency

Synopsis: Sets dependency at specified index

Declaration:
```
procedure SetDependency(Index: Integer; const AValue: TDependency)
```

Visibility: default

Description: Assigns a TDependency (988) object at the specified index position within the dependencies collection.

Updates the dependency entry with new configuration and settings for package build requirements.

29.24.6 TDependencies.Add

Synopsis: Adds a new package dependency to the collection

Declaration:
```
function Add(const Value: string) : TDependency
function Add(const Value: string; const OSes: TOSes) : TDependency
function Add(const Value: string; const CPUs: TCPUS; const OSes: TOSes)
             : TDependency
```

Visibility: public

Description: Creates and adds a new dependency with the specified package name to the collection. The dependency can be made platform-specific by providing CPU and OS restrictions.

Dependencies ensure that the required package is available before compilation of the dependent target begins.

29.24.7 TDependencies.AddUnit

Synopsis: Adds a unit dependency to the collection

Declaration:
```
function AddUnit(const Value: string) : TDependency
function AddUnit(const Value: string; const OSes: TOSes) : TDependency
function AddUnit(const Value: string; const CPUs: TCPUS;
                 const OSes: TOSes) : TDependency
```

Visibility: public

Description: Creates and adds a new dependency on a specific unit within the same package, so that the installer can decide to compile the dependency first when determining the order of compilation. The dependency can be made conditional by specifying CPU architectures and operating systems where this dependency applies.

See also: Add (987), AddInclude (988), depUnit (921)

29.24.8 TDependencies.AddInclude

Synopsis: Adds an include file dependency to the collection

Declaration:
```
function AddInclude(const Value: string) : TDependency
function AddInclude(const Value: string; const OSes: TOSes)
                    : TDependency
function AddInclude(const Value: string; const CPUs: TCPUS;
                    const OSes: TOSes) : TDependency
```

Visibility: public

Description: Creates and adds a new dependency on an include file. The dependency can be made conditional by specifying CPU architectures and operating systems where this dependency applies.

See also: Add (987), AddUnit (987), depInclude (921)

29.24.9 TDependencies.Dependencies

Synopsis: Indexed access to dependency objects in the collection

Declaration:
```
Property Dependencies[Index: Integer]: TDependency; default
```

Visibility: public

Access: Read,Write

Description: Provides indexed read and write access to TDependency (988) objects stored in the collection.

This default property enables array-style access to individual dependency entries for configuration and management.

29.25 TDependency

29.25.1 Description

`TDependency` represents a dependency relationship between build targets or packages. Dependencies can be on other packages, individual units, include files, or specific targets.

The dependency system ensures that required components are built before dependent targets, manages version requirements, and handles conditional dependencies based on CPU and OS targets.

Dependencies support version checking through the `Version` property and can be conditional on specific CPU architectures and operating systems inherited from TConditionalString (965).

See also: TDependencies (986), TDependencyType (921), TConditionalString (965), TPackage (1009), TTarget (1049)

29.25.2 Method overview

Page	Method	Description
989	Create	Creates a new package dependency instance
989	Destroy	Destroys the dependency object and releases resources

29.25.3 Property overview

Page	Properties	Access	Description
989	DependencyType	r	Type of dependency relationship
990	RequireChecksum	rw	Controls checksum verification for the dependency
989	Target	rw	Target name that this dependency references
990	TargetFileName	rw	Specific filename for the dependency target
990	Version	rw	Version requirement for the dependency

29.25.4 TDependency.Create

Synopsis: Creates a new package dependency instance

Declaration: `constructor Create;  Override`

Visibility: public

Description: `Create` initializes a new TDependency (988) object with the specified collection owner.

The constructor establishes the dependency relationship and registers it with the provided collection.

Dependencies are used to define build order requirements and package relationship constraints.

29.25.5 TDependency.Destroy

Synopsis: Destroys the dependency object and releases resources

Declaration: `destructor Destroy;  Override`

Visibility: public

Description: `Destroy` cleans up the dependency object and removes it from its containing collection.

The destructor ensures proper cleanup of version objects and conditional string resources.

The dependency is automatically unregistered from the build system when destroyed.

29.25.6 TDependency.Target

Synopsis: Target name that this dependency references

Declaration: `Property Target : TObject`

Visibility: public

Access: Read,Write

Description: Name of the target, unit, package, or include file that this dependency requires.

The target must be available for the dependent target to compile successfully.

29.25.7 TDependency.DependencyType

Synopsis: Type of dependency relationship

Declaration: `Property DependencyType : TDependencyType`

Visibility: public

Access: Read

Description: Specifies the type of dependency: package, implicit package, unit, or include file.

The dependency type determines how the dependency is resolved and linked.

This read-only property is set when the dependency is added to the dependencies collection.

See also: TDependencies.Add (987), TDependencies.AddUnit (987), TDependencies.AddInclude (988)

29.25.8 TDependency.TargetFileName

Synopsis: Specific filename for the dependency target

Declaration: `Property TargetFileName : string`

Visibility: public

Access: Read,Write

Description: Optional specific filename for the dependency target.

When specified, this overrides the default filename derived from the target name.

29.25.9 TDependency.Version

Synopsis: Version requirement for the dependency

Declaration: `Property Version : string`

Visibility: public

Access: Read,Write

Description: Version specification for the required dependency.

Used to ensure compatibility when multiple versions of a dependency are available.

29.25.10 TDependency.RequireChecksum

Synopsis: Controls checksum verification for the dependency

Declaration: `Property RequireChecksum : Cardinal`

Visibility: public

Access: Read,Write

Description: When `True`, requires checksum verification of the dependency file.

Ensures integrity and prevents use of corrupted or modified dependencies.

29.26 TDictionary

29.26.1 Description

`TDictionary` provides a string substitution system for variable and function expansion.

This class manages both static variable assignments and dynamic function callbacks for resolving configuration values.

The dictionary is used extensively throughout the build system to resolve paths, variables, and configuration parameters.

29.26.2 Method overview

Page	Method	Description
992	AddFunction	Adds a function replacement callback to the dictionary
991	AddVariable	Adds a variable name-value pair to the dictionary
991	Create	Creates a new string substitution dictionary
991	Destroy	Destroys the dictionary and releases resources
992	GetValue	Retrieves substituted value for dictionary entry
992	RemoveItem	Removes a variable or function from the dictionary
992	ReplaceStrings	Performs string substitution using dictionary variables
993	Substitute	Substitutes macros in a string using the dictionary

29.26.3 TDictionary.Create

Synopsis: Creates a new string substitution dictionary

Declaration: `constructor Create(AOwner: TComponent);  Override`

Visibility: public

Description: Creates a new `TDictionary` instance that can store variable name-value pairs and function callbacks for dynamic string substitution.

The dictionary is used throughout the build system to resolve configuration variables, paths, and dynamic values during compilation and installation.

29.26.4 TDictionary.Destroy

Synopsis: Destroys the dictionary and releases resources

Declaration: `destructor Destroy;  Override`

Visibility: public

Description: `Destroy` frees all memory used by the dictionary including variable storage and function callback lists.

This destructor ensures proper cleanup of all internal data structures and registered callbacks.

The destructor is called automatically when the dictionary object is destroyed or goes out of scope.

29.26.5 TDictionary.AddVariable

Synopsis: Adds a variable name-value pair to the dictionary

Declaration: `procedure AddVariable(const AName: string; const Value: string)`

Visibility: public

Description: Adds or updates a variable with the specified name and value in the dictionary. If a variable with the same name already exists, its value is replaced.

Variables are used for static string substitution in configuration files, command line arguments, and file paths.

29.26.6 TDictionary.AddFunction

Synopsis: Adds a function replacement callback to the dictionary

Declaration:
```
procedure AddFunction(const AName: string;
                      FReplacement: TReplaceFunction)
```

Visibility: public

Description: Adds a function callback with the specified name that provides dynamic value generation during string substitution. Functions can accept arguments and return computed values based on runtime conditions.

This enables advanced templating where values are calculated dynamically based on build parameters, system configuration, or package state.

29.26.7 TDictionary.RemoveItem

Synopsis: Removes a variable or function from the dictionary

Declaration:
```
procedure RemoveItem(const AName: string)
```

Visibility: public

Description: `RemoveItem` removes the specified variable or function callback from the dictionary.

After removal, attempts to resolve the removed item name will fail with an exception.

This method is useful for dynamically modifying the available substitution variables during build processes.

29.26.8 TDictionary.GetValue

Synopsis: Retrieves substituted value for dictionary entry

Declaration:
```
function GetValue(AName: string) : string
function GetValue(const AName: string; const Args: string) : string
                  ;  Virtual
```

Visibility: public

Description: Retrieves the value for the specified dictionary entry name. For variables, returns the stored string value. For functions, executes the callback with the provided arguments and returns the computed result.

Raises an exception if the requested entry name is not found in the dictionary.

29.26.9 TDictionary.ReplaceStrings

Synopsis: Performs string substitution using dictionary variables

Declaration:
```
function ReplaceStrings(const ASource: string; const MaxDepth: Integer)
                        : string;  Virtual
```

Visibility: public

Description: `ReplaceStrings` processes a source string and replaces all variable and function references with their resolved values.

The method supports nested substitutions up to the specified maximum depth to prevent infinite recursion.

Variable references use $(name) syntax, and function calls use $(name:arg1:arg2) format for parameter passing.

29.26.10 TDictionary.Substitute

Synopsis: Substitutes macros in a string using the dictionary

Declaration:
```
function Substitute(const Source: string; const Macros: Array of string)
                          : string;  Virtual
```

Visibility: public

Description: `Substitute` performs macro expansion and variable substitution on the input string using dictionary entries.

This method processes macro syntax patterns and replaces them with corresponding values from variables and function callbacks.

The substitution supports both simple variable expansion and complex function calls with parameter passing.

29.27 TFPCDefaults

29.27.1 Description

`TFPCDefaults` extends TCustomDefaults (967) with Free Pascal Compiler specific configuration settings and defaults. This class provides FPC-tailored build configuration management.

The class includes FPC-specific compiler paths, option handling, and build system integration optimized for Free Pascal development environments.

29.27.2 Method overview

Page	Method	Description
993	CompilerDefaults	Sets up FPC compiler default configuration

29.27.3 TFPCDefaults.CompilerDefaults

Synopsis: Sets up FPC compiler default configuration

Declaration:
```
procedure CompilerDefaults;  Override
```

Visibility: public

Description: `CompilerDefaults` initializes the FPC compiler with standard default settings and options. This method configures the compiler environment with appropriate paths, compilation modes, and target settings.

The setup includes standard FPC installation paths, library directories, and default compiler options suitable for typical Free Pascal development scenarios.

29.28 TFPCInstaller

29.28.1 Description

`TFPCInstaller` extends TCustomInstaller (983) with Free Pascal Compiler specific functionality and configuration. This class provides FPC-tailored installation and build management.

The FPC installer includes specific knowledge of FPC compiler behavior, standard library paths, and Free Pascal ecosystem conventions for package building and installation.

29.28.2 Method overview

Page	Method	Description
994	Create	Creates FPC installer instance

29.28.3 TFPCInstaller.Create

Synopsis: Creates FPC installer instance

Declaration: `constructor Create(AOwner: TComponent);  Override`

Visibility: public

Description: `Create` initializes a new `TFPCInstaller` instance with FPC-specific configuration and defaults. This constructor sets up the installer with Free Pascal Compiler specific build engine and settings.

The created installer is preconfigured for Free Pascal development with appropriate compiler paths, library directories, and build system integration.

This is the installer created when no installer class is passed to the Installer (936) function.

See also: Installer (936)

29.29 TfpmPlugin

29.29.1 Description

`TfpmPlugin` provides the foundation for implementing FPMake plugins that extend build system functionality. Plugins enable custom processing during various phases of the package building and installation process.

29.29.2 Method overview

Page	Method	Description
995	AfterResolvePackagePath	Called after resolving package path
995	BeforeResolvePackagePath	Called before resolving package path
995	ResolvePackagePath	Resolves package path location

29.29.3 Property overview

Page	Properties	Access	Description
995	Name	r	Name of the plugin

29.29.4 TfpmPlugin.BeforeResolvePackagePath

Synopsis: Called before resolving package path

Declaration:
```
procedure BeforeResolvePackagePath(ABuildEngine: TBuildEngine;
                                   APackage: TPackage;
                                   out AContinue: Boolean);  Virtual
```

Visibility: public

Description: `BeforeResolvePackagePath` is called by the build engine before resolving package paths during build operations. This method enables plugins to intercept and modify package path resolution logic.

Plugins can use this callback to implement custom package location strategies, add search paths, or modify the package discovery process according to specific requirements.

29.29.5 TfpmPlugin.ResolvePackagePath

Synopsis: Resolves package path location

Declaration:
```
procedure ResolvePackagePath(ABuildEngine: TBuildEngine;
                             APackage: TPackage;
                             SearchDirectory: string;
                             out AContinue: Boolean);  Virtual
```

Visibility: public

Description: `ResolvePackagePath` performs the actual package path resolution logic within the plugin. This method implements custom package discovery algorithms and location strategies.

The plugin can search in custom directories, apply naming conventions, or use alternative package discovery mechanisms to locate package files and directories.

29.29.6 TfpmPlugin.AfterResolvePackagePath

Synopsis: Called after resolving package path

Declaration:
```
procedure AfterResolvePackagePath(ABuildEngine: TBuildEngine;
                                  APackage: TPackage;
                                  out AContinue: Boolean);  Virtual
```

Visibility: public

Description: `AfterResolvePackagePath` is called by the build engine after package path resolution completes. This method enables plugins to perform post-resolution processing or cleanup operations.

29.29.7 TfpmPlugin.Name

Synopsis: Name of the plugin

Declaration:
```
Property Name : string
```

Visibility: public

Access: Read

Description: `Name` property holds the unique identifier for the plugin instance. This name is used for plugin registration, lookup, and identification within the build system.

Plugin names should be unique within the build environment.

29.30 TfpmPluginManager

29.30.1 Description

`TfpmPluginManager` manages the registration, instantiation, and lifecycle of FPMake plugins. This class provides centralized plugin management and coordinates plugin interactions with the build system.

The plugin manager handles plugin registration, maintains plugin collections, and provides the interface between the build engine and individual plugin instances.

29.30.2 Method overview

Page	Method	Description
997	AfterResolvePackagePath	Notifies all plugins after path resolution
996	BeforeResolvePackagePath	Notifies all plugins before path resolution
996	Destroy	Destroys plugin manager and registered plugins
996	RegisterPlugin	Registers plugin with the manager
997	ResolvePackagePath	Delegates package path resolution to plugins

29.30.3 TfpmPluginManager.Destroy

Synopsis: Destroys plugin manager and registered plugins

Declaration:
```
destructor Destroy;  Override
```

Visibility: public

Description: `Destroy` destroys the plugin manager and frees all registered plugin instances. This destructor ensures proper cleanup of all plugin resources and memory allocations.

The destructor unregisters all plugins, frees plugin instances, and releases any system resources held by the plugin management system.

29.30.4 TfpmPluginManager.RegisterPlugin

Synopsis: Registers plugin with the manager

Declaration:
```
procedure RegisterPlugin(APlugin: TfpmPluginClass)
```

Visibility: public

Description: `RegisterPlugin` adds a plugin instance to the manager's collection of active plugins. This method enables the plugin to participate in build system events and callbacks.

Registered plugins receive notifications during package resolution, compilation, and other build operations, allowing them to extend or customize build system behavior.

29.30.5 TfpmPluginManager.BeforeResolvePackagePath

Synopsis: Notifies all plugins before path resolution

Declaration:
```
procedure BeforeResolvePackagePath(ABuildEngine: TBuildEngine;
                                   APackage: TPackage;
                                   out AContinue: Boolean);  Override
```

Visibility: public

Description: `BeforeResolvePackagePath` notifies all registered plugins before package path resolution begins. This method allows plugins to prepare for or modify the upcoming path resolution process.

The manager iterates through all registered plugins, calling their `BeforeResolvePackagePath` methods to enable coordinated plugin processing during package discovery operations.

29.30.6 TfpmPluginManager.ResolvePackagePath

Synopsis: Delegates package path resolution to plugins

Declaration:
```
procedure ResolvePackagePath(ABuildEngine: TBuildEngine;
                             APackage: TPackage; SearchPath: string;
                             out AContinue: Boolean);  Override
```

Visibility: public

Description: `ResolvePackagePath` coordinates package path resolution by delegating to registered plugins. This method manages the plugin chain and handles resolution results from multiple plugin sources.

The manager calls each plugin's resolution method until a package is found or all plugins have been exhausted, enabling flexible and extensible package discovery mechanisms.

29.30.7 TfpmPluginManager.AfterResolvePackagePath

Synopsis: Notifies all plugins after path resolution

Declaration:
```
procedure AfterResolvePackagePath(ABuildEngine: TBuildEngine;
                                  APackage: TPackage;
                                  out AContinue: Boolean);  Override
```

Visibility: public

Description: `AfterResolvePackagePath` notifies all registered plugins after package path resolution completes. This method enables plugins to perform post-resolution processing or cleanup operations.

The manager provides resolution results to all plugins, allowing them to cache information, validate results, or trigger additional processing based on the resolved package paths.

29.31 TfpmResolvePackagePathsPlugin

29.31.1 Description

`TfpmResolvePackagePathsPlugin` implements a specialized plugin for package file path resolution. This plugin provides specific logic for locating package files and directories within the file system.

29.31.2 Method overview

Page	Method	Description
998	BeforeResolvePackagePath	Called before path resolution by plugin
998	ResolvePackagePath	Resolves package path using plugin logic

29.31.3 TfpmResolvePackagePathsPlugin.BeforeResolvePackagePath

Synopsis: Called before path resolution by plugin

Declaration:
```
procedure BeforeResolvePackagePath(ABuildEngine: TBuildEngine;
                                   APackage: TPackage;
                                   out AContinue: Boolean);  Override
```

Visibility: public

Description: `BeforeResolvePackagePath` is called before the plugin performs package path resolution. This method enables pre-processing and preparation for the package discovery operation.

29.31.4 TfpmResolvePackagePathsPlugin.ResolvePackagePath

Synopsis: Resolves package path using plugin logic

Declaration:
```
procedure ResolvePackagePath(ABuildEngine: TBuildEngine;
                             APackage: TPackage; SearchPath: string;
                             out AContinue: Boolean);  Override
```

Visibility: public

Description: `ResolvePackagePath` implements the core package path resolution logic for this plugin. This method searches for package files using plugin-specific strategies and search algorithms.

29.32 TFPVersion

29.32.1 Description

Class that encapsulates a Free Pascal version number with major, minor, micro, and build components. Provides methods for version comparison, parsing from strings, and formatting to strings.

29.32.2 Method overview

Page	Method	Description
999	Assign	Assign version from another source
999	Clear	Clear the version number to empty state
999	CompareVersion	Compare this version with another version
999	SameVersion	Check if versions are identical

29.32.3 Property overview

Page	Properties	Access	Description
1000	AsString	rw	Version number as formatted string
1001	Build	rw	Build version number
1000	Empty	r	Whether the version number is empty
1000	Major	rw	Major version number
1001	Micro	rw	Micro version number
1000	Minor	rw	Minor version number

29.32.4 TFPVersion.Clear

Synopsis: Clear the version number to empty state

Declaration: `procedure Clear`

Visibility: public

Description: Procedure that resets all version number components (major, minor, micro, build) to zero. Returns the version object to an empty, uninitialized state.

29.32.5 TFPVersion.Assign

Synopsis: Assign version from another source

Declaration: `procedure Assign(Source: TPersistent);  Override`

Visibility: public

Description: Procedure that copies version information from another TFPVersion (998) object or compatible source. Allows cloning or copying version data between version objects.

29.32.6 TFPVersion.CompareVersion

Synopsis: Compare this version with another version

Declaration: `function CompareVersion(AVersion: TFPVersion) : Integer`

Visibility: public

Description: `CompareVersion` performs a numerical comparison between this version object and another version.

The comparison follows semantic versioning rules, comparing major, minor, micro, and build numbers in sequence.

Returns a negative value if this version is older, zero if versions are equal, or positive if this version is newer.

29.32.7 TFPVersion.SameVersion

Synopsis: Check if versions are identical

Declaration: `function SameVersion(AVersion: TFPVersion) : Boolean`

Visibility: public

Description: `SameVersion` returns `True` if this version is exactly identical to the specified version.

This method performs an exact comparison of all version components including major, minor, micro, and build numbers.

Unlike CompareVersion (999), this method returns a boolean result rather than a comparison integer.

29.32.8 TFPVersion.AsString

Synopsis: Version number as formatted string

Declaration: `Property AsString : string`

Visibility: public

Access: Read,Write

Description: `AsString` returns the version number formatted as a human-readable string in the format "major.minor.micro.build".

Empty version components are omitted from the string representation, so version 1.2.0.0 appears as "1.2".

This property is commonly used for display purposes and version comparison logging.

29.32.9 TFPVersion.Empty

Synopsis: Whether the version number is empty

Declaration: `Property Empty : Boolean`

Visibility: public

Access: Read

Description: `Empty` returns `True` if all version number components (major, minor, micro, build) are zero.

This property is useful for checking whether a version has been properly initialized or represents a valid version number.

An empty version typically indicates an unset or invalid version state.

29.32.10 TFPVersion.Major

Synopsis: Major version number

Declaration: `Property Major : Integer`

Visibility: published

Access: Read,Write

Description: `Major` specifies the major version component, representing significant API changes or major releases.

In semantic versioning, major version increments indicate breaking changes or incompatible API modifications.

This value must be a non-negative integer and is the most significant component in version comparison.

29.32.11 TFPVersion.Minor

Synopsis: Minor version number

Declaration: `Property Minor : Integer`

Visibility: published

Access: Read,Write

Description: `Minor` represents the minor version component, indicating backwards-compatible functionality additions.

In semantic versioning, minor version increments signal new features that maintain API compatibility.

This value must be a non-negative integer and is compared after major version in version ordering.

29.32.12 TFPVersion.Micro

Synopsis: Micro version number

Declaration: `Property Micro : Integer`

Visibility: published

Access: Read,Write

Description: `Micro` represents the micro (or patch) version component for backwards-compatible bug fixes.

In semantic versioning, micro version increments indicate bug fixes and small improvements without API changes.

This value must be a non-negative integer and is the least significant component in standard version comparison.

29.32.13 TFPVersion.Build

Synopsis: Build version number

Declaration: `Property Build : Integer`

Visibility: published

Access: Read,Write

Description: `Build` represents the build version component for distinguishing different builds of the same version.

Build numbers are typically used for internal tracking, nightly builds, or continuous integration systems.

This value must be a non-negative integer and is usually the least significant component in extended version comparison.

29.33 TFunctionItem

29.33.1 Description

`TFunctionItem` represents a dictionary entry that stores a function callback with an associated key.

Function items provide extensibility by allowing custom functions to be registered and called during template processing and build configuration evaluation.

29.33.2 Method overview

Page	Method	Description
1002	Create	Creates function item with specified callback

29.33.3 TFunctionItem.Create

Synopsis: Creates function item with specified callback

Declaration: `constructor Create(AFunc: TReplaceFunction)`

Visibility: default

Description: `Create` initializes a new `TFunctionItem` instance with the specified function callback. This constructor stores the provided function reference for later invocation during template evaluation.

29.34 TNamedCollection

29.34.1 Description

Generic collection class that manages a list of TNamedItem (1003) objects. Provides name-based lookup functionality and ensures unique naming within the collection.

29.34.2 Method overview

Page	Method	Description
1002	IndexOfName	Find index of item by name
1002	ItemByName	Get item by name

29.34.3 Property overview

Page	Properties	Access	Description
1003	Items	rw	Indexed access to collection items
1003	UniqueNames	r	Controls whether item names must be unique

29.34.4 TNamedCollection.IndexOfName

Synopsis: Find index of item by name

Declaration: `function IndexOfName(const AName: string) : Integer`

Visibility: public

Description: Searches for an item with the specified name in the collection and returns its zero-based index. Returns -1 if no item with the given name is found.

This provides efficient name-based lookup for finding specific items in named collections.

29.34.5 TNamedCollection.ItemByName

Synopsis: Get item by name

Declaration: `function ItemByName(const AName: string) : TNamedItem`

Visibility: public

Description: Retrieves an item from the collection by its name. Returns the TNamedItem (1003) object if found, or `nil` if no item with the specified name exists.

29.34.6 TNamedCollection.UniqueNames

Synopsis: Controls whether item names must be unique

Declaration: `Property UniqueNames : Boolean`

Visibility: public

Access: Read

Description: Boolean property that determines whether the collection enforces unique naming. When `True`, attempting to add an item with a duplicate name will raise an exception.

29.34.7 TNamedCollection.Items

Synopsis: Indexed access to collection items

Declaration: `Property Items[Index: Integer]: TNamedItem`

Visibility: public

Access: Read,Write

Description: Indexed property providing direct access to collection items by their zero-based index. Allows reading and writing of TNamedItem (1003) objects at specific positions.

29.35 TNamedItem

29.35.1 Description

Abstract base class that provides a `Name` property for objects that need to be identified by name. Serves as the foundation for many fpmake objects that require named identification.

29.35.2 Property overview

Page	Properties	Access	Description
1003	Name	rw	Name of the item

29.35.3 TNamedItem.Name

Synopsis: Name of the item

Declaration: `Property Name : string`

Visibility: public

Access: Read,Write

Description: String property that stores the unique name identifier for this item. Used for lookups, references, and display purposes throughout the build system.

29.36 TNamedItemList

29.36.1 Description

Specialized list class that extends basic list functionality with name-based lookup capabilities. Provides efficient access to TNamedItem (1003) objects by both index and name.

29.36.2 Method overview

Page	Method	Description
1004	IndexOfName	Find index of named item
1004	ItemByName	Get named item by name

29.36.3 Property overview

Page	Properties	Access	Description
1004	NamedItems	rw	Indexed access to named items

29.36.4 TNamedItemList.IndexOfName

Synopsis: Find index of named item

Declaration: `function IndexOfName(const AName: string) : Integer`

Visibility: public

Description: Searches for an item with the specified name and returns its zero-based index in the list. Returns -1 if no item with the given name is found.

29.36.5 TNamedItemList.ItemByName

Synopsis: Get named item by name

Declaration: `function ItemByName(const ANAme: string) : TNamedItem`

Visibility: public

Description: Retrieves an item from the list by its name. Returns the TNamedItem (1003) object if found, or `nil` if no item with the specified name exists.

29.36.6 TNamedItemList.NamedItems

Synopsis: Indexed access to named items

Declaration: `Property NamedItems[Index: Integer]: TNamedItem; default`

Visibility: public

Access: Read,Write

Description: Property providing indexed access to TNamedItem (1003) objects by their name. Allows retrieving items using string names as array indices.

29.37 TNotifyEventCollection

29.37.1 Description

`TNotifyEventCollection` is a specialized collection that manages multiple notification event handlers.

This class provides methods for adding, removing, and invoking notification events at specific build process stages.

Build systems use this collection to maintain lists of callbacks that should be executed at defined trigger points.

29.37.2 Method overview

Page	Method	Description
1005	AppendEvent	Adds a notification event handler to the collection
1006	AppendProcEvent	Adds a procedural notification event to the collection
1006	CallEvents	Invokes all event handlers for the specified action
1005	create	Creates a new notification event collection
1005	GetEnumerator	Gets enumerator for iterating notify events

29.37.3 TNotifyEventCollection.create

Synopsis: Creates a new notification event collection

Declaration:
```
constructor create(ASupportedActionSet: TNotifyEventActionSet)
```

Visibility: public

Description: `create` initializes a new TNotifyEventCollection (1005) instance.

The constructor sets up an empty collection ready to accept notification event items.

The collection is properly initialized with all necessary internal structures for event management.

29.37.4 TNotifyEventCollection.GetEnumerator

Synopsis: Gets enumerator for iterating notify events

Declaration:
```
function GetEnumerator : TNotifyEventEnumerator
```

Visibility: public

Description: `GetEnumerator` returns a TNotifyEventEnumerator (1006) for iterating through the collection.

This method enables for-in loop support for notification event collections.

The enumerator provides sequential access to each notification event handler in the collection.

29.37.5 TNotifyEventCollection.AppendEvent

Synopsis: Adds a notification event handler to the collection

Declaration:
```
procedure AppendEvent(AnAction: TNotifyEventAction;
                      AnEvent: TNotifyEvent)
```

Visibility: public

Description: `AppendEvent` adds a new notification event handler to the collection for a specific action type.

The event handler will be triggered when the specified build action occurs during package processing.

Multiple handlers can be associated with the same action type and will be executed in sequence.

29.37.6 TNotifyEventCollection.AppendProcEvent

Synopsis: Adds a procedural notification event to the collection

Declaration:
```
procedure AppendProcEvent(AnACtion: TNotifyEventAction;
                          AnProcEvent: TNotifyProcEvent)
```

Visibility: public

Description: `AppendProcEvent` adds a procedural notification event handler to the collection for a specific action type.

This method accepts function pointers rather than object method pointers, enabling procedural callback support.

The procedural event handler will be triggered when the specified build action occurs during package processing.

29.37.7 TNotifyEventCollection.CallEvents

Synopsis: Invokes all event handlers for the specified action

Declaration:
```
procedure CallEvents(AnAction: TNotifyEventAction; Sender: TObject)
```

Visibility: public

Description: `CallEvents` executes all registered notification event handlers that match the specified action type.

This method is called internally by the build system when specific build actions occur.

All matching event handlers are invoked in sequence with the provided sender object as parameter.

29.38 TNotifyEventEnumerator

29.38.1 Description

`TNotifyEventEnumerator` provides iteration support for collections of notification events.

This class enables for-in loop syntax when working with notification event collections.

The enumerator maintains current position state and provides access to individual event items during iteration.

29.38.2 Method overview

Page	Method	Description
1007	GetCurrent	Gets current notify event in enumeration

29.38.3 Property overview

Page	Properties	Access	Description
1007	Current	r	Current notification event in the enumeration

29.38.4 TNotifyEventEnumerator.GetCurrent

Synopsis: Gets current notify event in enumeration

Declaration: `function GetCurrent : TNotifyEventItem`

Visibility: public

Description: `GetCurrent` returns the current notification event procedure at the enumerator's position.

This method provides access to the notification event handler currently pointed to by the enumerator.

The method is typically called internally during for-in loop iteration to retrieve the current item.

29.38.5 TNotifyEventEnumerator.Current

Synopsis: Current notification event in the enumeration

Declaration: `Property Current : TNotifyEventItem`

Visibility: public

Access: Read

Description: `Current` returns the current notification event procedure at the enumerator's position.

This property provides convenient access to the notification event handler currently pointed to during iteration.

The property is used in for-in loops to access each event handler as the enumeration progresses.

29.39 TNotifyEventItem

29.39.1 Description

`TNotifyEventItem` represents a single notification event configuration that associates callback handlers with specific build actions.

This class encapsulates the event trigger conditions and the corresponding callback procedures or functions.

Build systems use these items to implement custom behavior at defined points in the build process.

29.39.2 Method overview

Page	Method	Description
1008	CallEvent	Invokes the notification event callback

29.39.3 Property overview

Page	Properties	Access	Description
1008	OnAction	rw	Action type that triggers this notification event
1008	OnEvent	rw	Standard notify event callback procedure
1008	OnProcEvent	rw	Procedural notify event callback

29.39.4 TNotifyEventItem.CallEvent

Synopsis: Invokes the notification event callback

Declaration: `procedure CallEvent(Sender: TObject)`

Visibility: public

Description: `CallEvent` executes the configured notification event callback with the provided sender object.

This method checks which callback type is assigned and calls the appropriate handler procedure or method.

The method is called internally by the build system when the associated action trigger occurs.

29.39.5 TNotifyEventItem.OnAction

Synopsis: Action type that triggers this notification event

Declaration: `Property OnAction : TNotifyEventAction`

Visibility: public

Access: Read,Write

Description: `OnAction` specifies the TNotifyEventAction (924) that will trigger this notification event.

When the build process reaches the specified action point, the associated callback handlers will be executed.

This property determines at which stage of the build process the event will fire.

29.39.6 TNotifyEventItem.OnEvent

Synopsis: Standard notify event callback procedure

Declaration: `Property OnEvent : TNotifyEvent`

Visibility: public

Access: Read,Write

Description: `OnEvent` contains a standard notification event handler that follows the `TNotifyEvent` signature.

This property accepts method pointers that receive a sender object as parameter when the event fires.

Use this property for object-oriented event handling with standard notification event signatures.

29.39.7 TNotifyEventItem.OnProcEvent

Synopsis: Procedural notify event callback

Declaration: `Property OnProcEvent : TNotifyProcEvent`

Visibility: public

Access: Read,Write

Description: `OnProcEvent` contains a procedural notification event handler for function pointer-based callbacks.

This property accepts procedure pointers that can be called when the specified build action occurs.

Use this property for procedural event handling without object context requirements.

29.40 TPackage

29.40.1 Description

`TPackage` is the central class in the fpmake build system that represents a complete package. A package contains build targets, dependencies, installation files, and all configuration needed for compilation and deployment.

Each package can contain multiple targets such as units, programs, and libraries. The package manages compilation options, search paths, dependencies on other packages, and installation procedures. It supports cross-platform compilation with CPU and OS-specific settings.

It handles the following aspects:

- Management of TTarget (1049) build targets
- Dependency tracking through TDependency (988) objects
- Cross-platform compilation support
- Package variants for different build configurations
- Custom installation and archiving procedures

See also: TTarget (1049), TTargets (1061), TDependency (988), TPackageVariants (1039), TBuildEngine (941), TCustomInstaller (983)

29.40.2 Method overview

Page	Method	Description
1017	AddPackageVariant	Adds a package variant to the package
1017	ApplyPackageVariantToCompilerOptions	Applies package variant settings to compiler options
1013	Create	Creates a new package instance
1013	destroy	Destroys the package instance and frees resources
1018	EnterResolveDirsCS	Enters critical section for directory resolution
1016	GetArchiveFiles	Gets list of files to include in package archive
1016	GetArchiveSourceFiles	Gets list of source files for package archive
1015	GetBinOutputDir	Returns the output directory for binary files
1015	GetCleanFiles	Gets list of files to be cleaned for the package
1015	GetInstallFiles	Gets list of files to be installed for the package
1015	GetInstallSourceFiles	Gets list of source files to be installed
1016	GetManifest	Gets package manifest information
1014	GetPackageUnitInstallDir	Gets installation directory for package units
1014	GetPackageVariantsByName	Gets package variants collection by name
1014	GetUnitConfigOutputFilename	Gets output filename for unit configuration file
1013	GetUnitsOutputDir	Returns the output directory for compiled units
1013	HaveOptions	Checks if package has global compilation options
1014	InheritPackageVariantsFromDependency	Inherits package variants from a dependency package
1018	LeaveResolveDirsCS	Leaves critical section for directory resolution
1016	ListPackage	Lists package information to a string list
1017	LoadUnitConfigFromFile	Loads unit configuration from a file
1017	SaveUnitConfigToFile	Saves unit configuration to a file
1017	SetDefaultPackageVariant	Sets the default package variant for the package
1015	SetPackageUnitInstallDir	Sets installation directory for package units
1014	SetUnitsOutputDir	Sets the output directory for compiled units

29.40.3 Property overview

Page	Properties	Access	Description
1031	AfterArchive	rw	Event handler executed after package archiving
1031	AfterArchiveProc	rw	Procedural event handler after archiving
1030	AfterClean	rw	Event handler executed after package cleanup
1030	AfterCleanProc	rw	Procedural event handler after cleanup
1028	AfterCompile	rw	Event handler executed after package compilation
1028	AfterCompileProc	rw	Procedural event handler after compilation
1029	AfterInstall	rw	Event handler executed after package installation
1029	AfterInstallProc	rw	Procedural event handler after installation
1032	AfterManifest	rw	Event handler executed after manifest creation
1032	AfterManifestProc	rw	Procedural event handler after manifest creation
1033	AfterPkgList	rw	Event handler executed after package listing
1033	AfterPkgListProc	rw	Procedural event handler after package listing
1020	Author	rw	Author or maintainer of the package
1031	BeforeArchive	rw	Event handler executed before package archiving
1031	BeforeArchiveProc	rw	Procedural event handler before archiving
1030	BeforeClean	rw	Event handler executed before package cleanup
1030	BeforeCleanProc	rw	Procedural event handler before cleanup
1028	BeforeCompile	rw	Event handler executed before package compilation
1028	BeforeCompileProc	rw	Procedural event handler before compilation
1029	BeforeInstall	rw	Event handler executed before package installation
1029	BeforeInstallProc	rw	Procedural event handler before installation
1032	BeforeManifest	rw	Event handler executed before manifest creation
1032	BeforeManifestProc	rw	Procedural event handler before manifest creation
1033	BeforePkgList	rw	Event handler executed before package listing
1033	BeforePkgListProc	rw	Procedural event handler before package listing
1022	BuildMode	r	Default build mode configuration for the package
1026	CleanFiles	r	Collection of files to be cleaned
1026	Commands	r	Collection of custom build commands
1023	CPUs	rw	Supported CPU architectures for the package
1026	Dependencies	r	Collection of package dependencies
1020	Description	rw	Descriptive text about the package
1021	DescriptionFile	rw	File containing package description text
1020	Directory	rw	Base directory path of the package
1019	DownloadURL	rw	URL for downloading the package
1019	Email	rw	Contact email address for the package
1025	ExamplePath	r	Search paths for example files
1018	FileName	rw	Filename of the package definition file
1022	Flags	r	Package flags and behavior settings
1025	FPDocFormat	rw	Output format for FPDoc documentation
1019	HomepageURL	rw	URL of the package homepage
1024	IncludePath	r	Search paths for include files
1021	InstalledChecksum	rw	Checksum of installed package files
1025	InstallFiles	r	Collection of files to be installed
1021	IsFPMakeAddIn	rw	Whether this package is an FPMake add-in
1020	License	rw	License terms for the package
1023	NeedLibC	rw	Whether package requires C library linkage
1024	ObjectPath	r	Search paths for object files
1023	Options	rw	Compiler options for the package
1023	OSes	rw	Supported operating systems for the package
1022	PackageVersion	r	Version information for the package
1019	ShortName	rw	Short name identifier for the package
1024	SourcePath	r	Search paths for source files
1027	Sources	r	Collection of source files
1026	State	rw	Current build state of the package
1021	SupportBuildModes	rw	Whether package supports different build modes
1027	Targets	r	Collection of build targets
1025	TestPath	r	Search paths for test files
1022	TransmitOptions	rw	Options transmitted to dependent packages
1027	UnitConfigFileName	rw	Filename for unit configuration file

29.40.4 TPackage.Create

Synopsis: Creates a new package instance

Declaration: `constructor Create(ACollection: TCollection);  Override`

Visibility: public

Description: Creates a new `TPackage` instance and initializes all internal collections and properties. The constructor sets up default values and creates collections for targets, dependencies, and other package components.

See also: destroy (1013)

29.40.5 TPackage.destroy

Synopsis: Destroys the package instance and frees resources

Declaration: `destructor destroy;  Override`

Visibility: public

Description: Destroys the `TPackage` instance and frees all associated resources including target collections, dependency lists, and internal data structures.

See also: Create (1013)

29.40.6 TPackage.HaveOptions

Synopsis: Checks if package has global compilation options

Declaration: `function HaveOptions : Boolean`

Visibility: public

Description: Returns `True` if the package has global compiler options defined, `False` otherwise.

These options apply to all targets in the package unless overridden.

29.40.7 TPackage.GetUnitsOutputDir

Synopsis: Returns the output directory for compiled units

Declaration: `function GetUnitsOutputDir(ACPU: TCpu; AOS: TOS) : string`

Visibility: public

Description: Returns the directory where compiled unit files (.ppu) will be placed for the specified CPU, OS, and subtarget combination.

The directory path is constructed based on package configuration and platform settings.

29.40.8 TPackage.GetUnitConfigOutputFilename

Synopsis: Gets output filename for unit configuration file

Declaration: `function GetUnitConfigOutputFilename(ACPU: TCpu; AOS: TOS) : string`

Visibility: public

Description: Returns the filename for the unit configuration file (.fpc) that contains build settings and unit paths for the specified CPU, OS, and subtarget combination. This file is used by the compiler to locate units and configure compilation.

29.40.9 TPackage.InheritPackageVariantsFromDependency

Synopsis: Inherits package variants from a dependency package

Declaration:
```
procedure InheritPackageVariantsFromDependency
                                                    (ADependencyPackage: TPackage)
```

Visibility: public

Description: Copies package variants from a dependency package into this package, allowing build configurations to be inherited through the dependency chain. This ensures consistent variant settings across dependent packages.

29.40.10 TPackage.GetPackageVariantsByName

Synopsis: Gets package variants collection by name

Declaration: `function GetPackageVariantsByName(AName: string) : TPackageVariants`

Visibility: public

Description: Searches for and returns a package variants collection with the specified name. Returns `nil` if no variants collection with the given name is found.

29.40.11 TPackage.SetUnitsOutputDir

Synopsis: Sets the output directory for compiled units

Declaration: `procedure SetUnitsOutputDir(AValue: string)`

Visibility: public

Description: Sets the base directory where compiled unit files (.ppu) will be stored for this package. The actual output location may include subdirectories based on CPU and OS target settings.

29.40.12 TPackage.GetPackageUnitInstallDir

Synopsis: Gets installation directory for package units

Declaration: `function GetPackageUnitInstallDir(ACPU: TCpu; AOS: TOS) : string`

Visibility: public

Description: Returns the directory path where compiled unit files for this package will be installed for the specified CPU and OS combination. This is where other packages will look for this package's units.

29.40.13 TPackage.SetPackageUnitInstallDir

Synopsis: Sets installation directory for package units

Declaration:
```
procedure SetPackageUnitInstallDir(AValue: string)
```

Visibility: public

Description: Sets the directory where compiled unit files for this package will be installed. This directory is where other packages will search for this package's compiled units during their own build process.

29.40.14 TPackage.GetBinOutputDir

Synopsis: Returns the output directory for binary files

Declaration:
```
function GetBinOutputDir(ACPU: TCpu; AOS: TOS) : string
```

Visibility: public

Description: Returns the directory where binary files (executables, libraries) will be placed for the specified CPU, OS, and subtarget combination.

Binary files include programs and shared libraries generated by the build process.

29.40.15 TPackage.GetCleanFiles

Synopsis: Gets list of files to be cleaned for the package

Declaration:
```
procedure GetCleanFiles(List: TStrings; ACPU: TCpu; AOS: TOS);  Virtual
```

Visibility: public

Description: Populates the provided list with filenames of all generated files that should be removed during package cleanup. This includes compiled units, object files, executables, and other build artifacts for the specified target platform.

29.40.16 TPackage.GetInstallFiles

Synopsis: Gets list of files to be installed for the package

Declaration:
```
procedure GetInstallFiles(List: TStrings; Types: TTargetTypes;
                          ACPU: TCpu; AOS: TOS);  Virtual
```

Visibility: public

Description: Populates the provided list with files that should be installed when deploying this package. The file types parameter controls which categories of files are included in the installation list.

29.40.17 TPackage.GetInstallSourceFiles

Synopsis: Gets list of source files to be installed

Declaration:
```
procedure GetInstallSourceFiles(List: TStrings;
                                SourceTypes: TSourceTypes;
                                TargetTypes: TTargetTypes);  Virtual
```

Visibility: public

Description: Populates the provided list with source files (.pp, .pas, .inc) that should be installed for developer access. This allows users to examine and modify the package source code.

29.40.18 TPackage.GetArchiveFiles

Synopsis: Gets list of files to include in package archive

Declaration:
```
procedure GetArchiveFiles(List: TStrings; ACPU: TCpu; AOS: TOS)
                          ;  Virtual
```

Visibility: public

Description: Populates the provided list with filenames of all files that should be included when creating a package archive. This includes source files, documentation, examples, and other distributable content for the specified target platform.

29.40.19 TPackage.GetArchiveSourceFiles

Synopsis: Gets list of source files for package archive

Declaration:
```
procedure GetArchiveSourceFiles(List: TStrings);  Virtual
```

Visibility: public

Description: Populates the provided list with source files that should be included when creating a distributable package archive. This includes Pascal source files, include files, and other source-related content for the specified target platform.

29.40.20 TPackage.GetManifest

Synopsis: Gets package manifest information

Declaration:
```
procedure GetManifest(Manifest: TStrings)
```

Visibility: public

Description: `GetManifest` populates the provided manifest object with information about the package. This includes package metadata, version details, dependencies, targets, and other package configuration data.

The manifest contains structured information that can be used for package registration, dependency resolution, and package management operations within the fpmake system.

29.40.21 TPackage.ListPackage

Synopsis: Lists package information to a string list

Declaration:
```
procedure ListPackage(PkgList: TStrings)
```

Visibility: public

Description: Populates the provided string list with detailed information about this package including name, version, description, targets, dependencies, and other configuration details for diagnostic or reporting purposes.

29.40.22 TPackage.AddPackageVariant

Synopsis: Adds a package variant to the package

Declaration:
```
procedure AddPackageVariant(APackageVariant: TPackageVariants)
```

Visibility: public

Description: Adds an existing package variant object to this package, making it available for use during builds. The variant provides alternative build configurations with different compiler options and settings.

29.40.23 TPackage.ApplyPackageVariantToCompilerOptions

Synopsis: Applies package variant settings to compiler options

Declaration:
```
procedure ApplyPackageVariantToCompilerOptions
                                                    (ACompilerOptions: TStrings)
```

Visibility: public

Description: `ApplyPackageVariantToCompilerOptions` modifies the provided compiler options by applying settings from the package variant configuration. Package variants allow customizing compilation parameters for different build scenarios or target configurations.

This method ensures that variant-specific compiler flags, include paths, and other compilation settings are properly applied to the build process.

29.40.24 TPackage.SetDefaultPackageVariant

Synopsis: Sets the default package variant for the package

Declaration:
```
procedure SetDefaultPackageVariant
```

Visibility: public

Description: Sets the package variant that will be used when no specific variant is requested during the build process. This ensures consistent default behavior for package compilation and installation.

29.40.25 TPackage.LoadUnitConfigFromFile

Synopsis: Loads unit configuration from a file

Declaration:
```
procedure LoadUnitConfigFromFile(const AFileName: string)
```

Visibility: public

Description: Loads unit search paths and configuration settings from an existing unit configuration file (.fpc). This allows restoration of previously saved compiler configuration for consistent builds.

29.40.26 TPackage.SaveUnitConfigToFile

Synopsis: Saves unit configuration to a file

Declaration:
```
procedure SaveUnitConfigToFile(const AFileName: string; ACPU: TCpu;
                                    AOS: TOS)
```

Visibility: public

Description: Saves current unit search paths and configuration settings to a unit configuration file (.fpc) for the specified CPU and OS combination. This file can later be used by the compiler for consistent build settings.

29.40.27 TPackage.EnterResolveDirsCS

Synopsis: Enters critical section for directory resolution

Declaration: `procedure EnterResolveDirsCS`

Visibility: public

Description: Enters a critical section to ensure thread-safe access during directory path resolution operations. This prevents race conditions when multiple threads are resolving directory paths simultaneously.

29.40.28 TPackage.LeaveResolveDirsCS

Synopsis: Leaves critical section for directory resolution

Declaration: `procedure LeaveResolveDirsCS`

Visibility: public

Description: Exits the critical section for directory path resolution operations, allowing other threads to access directory resolution functions. Must be called after every call to `EnterResolveDirsCS`.

29.40.29 TPackage.Version

Synopsis: Version string of the package

Declaration: `Property Version : string`

Visibility: public

Access: Read,Write

Description: The `Version` property contains the version string of the package. This version is used for dependency checking and package identification.

See also: TFPVersion (998), PackageVersion (1022)

29.40.30 TPackage.FileName

Synopsis: Filename of the package definition file

Declaration: `Property FileName : string`

Visibility: public

Access: Read,Write

Description: `FileName` contains the full path to the package definition file (typically fpmake.pp) that describes this package's configuration, targets, and dependencies.

29.40.31 TPackage.ShortName

Synopsis: Short name identifier for the package

Declaration: `Property ShortName : string`

Visibility: public

Access: Read,Write

Description: `ShortName` provides a concise identifier for the package, typically used in package listings and references. This is usually a brief version of the full package name.

29.40.32 TPackage.HomepageURL

Synopsis: URL of the package homepage

Declaration: `Property HomepageURL : string`

Visibility: public

Access: Read,Write

Description: `HomepageURL` contains the web address of the package's official homepage or project site. This provides users with access to documentation, news, and additional information about the package.

29.40.33 TPackage.DownloadURL

Synopsis: URL for downloading the package

Declaration: `Property DownloadURL : string`

Visibility: public

Access: Read,Write

Description: `DownloadURL` specifies the web address where the package can be downloaded. This is used by package management systems to retrieve the package source code or binaries.

29.40.34 TPackage.Email

Synopsis: Contact email address for the package

Declaration: `Property Email : string`

Visibility: public

Access: Read,Write

Description: `Email` contains the contact email address for the package maintainer or author. This provides a way for users to report issues, ask questions, or contribute to the package.

29.40.35 TPackage.Author

Synopsis: Author or maintainer of the package

Declaration: `Property Author : string`

Visibility: public

Access: Read,Write

Description: `Author` contains the name of the person or organization responsible for creating and maintaining the package. This provides attribution and contact information for the package.

29.40.36 TPackage.License

Synopsis: License terms for the package

Declaration: `Property License : string`

Visibility: public

Access: Read,Write

Description: `License` specifies the legal license terms under which the package is distributed. This information helps users understand their rights and obligations when using the package.

29.40.37 TPackage.Directory

Synopsis: Base directory path of the package

Declaration: `Property Directory : string`

Visibility: public

Access: Read,Write

Description: The `Directory` property specifies the base directory where the package files are located. All relative paths within the package are resolved relative to this directory.

See also: FileName (1018)

29.40.38 TPackage.Description

Synopsis: Descriptive text about the package

Declaration: `Property Description : string`

Visibility: public

Access: Read,Write

Description: The `Description` property contains a human-readable description of the package. This description can be loaded from a file specified in DescriptionFile (1021) or set directly.

See also: DescriptionFile (1021)

29.40.39 TPackage.DescriptionFile

Synopsis: File containing package description text

Declaration: `Property DescriptionFile : string`

Visibility: public

Access: Read,Write

Description: `DescriptionFile` specifies the path to a text file containing a detailed description of the package. When set, the file contents are loaded into the `Description` property.

29.40.40 TPackage.InstalledChecksum

Synopsis: Checksum of installed package files

Declaration: `Property InstalledChecksum : Cardinal`

Visibility: public

Access: Read,Write

Description: `InstalledChecksum` contains a checksum value of the installed package files. This is used to verify package integrity and detect modifications after installation.

29.40.41 TPackage.IsFPMakeAddIn

Synopsis: Whether this package is an FPMake add-in

Declaration: `Property IsFPMakeAddIn : Boolean`

Visibility: public

Access: Read,Write

Description: `IsFPMakeAddIn` indicates whether this package provides additional functionality for the FPMake build system itself. Add-in packages extend FPMake with custom build commands and capabilities.

29.40.42 TPackage.SupportBuildModes

Synopsis: Whether package supports different build modes

Declaration: `Property SupportBuildModes : TBuildModes`

Visibility: public

Access: Read,Write

Description: `SupportBuildModes` determines which build modes the installer can use when compiling this package: using a build unit or one by one, or both.

See also: BuildMode (1022), TCustomDefaults.BuildMode (981)

29.40.43 TPackage.BuildMode

Synopsis: Default build mode configuration for the package

Declaration: `Property BuildMode : TBuildMode`

Visibility: public

Access: Read

Description: `BuildMode` specifies the default build mode configuration for the package, determining whether the installer will compile using a build unit or one by one. Note that the build mode should be in the set of suppored modes SupportBuildModes (1021).

See also: SupportBuildModes (1021), TCustomDefaults.BuildMode (981)

29.40.44 TPackage.Flags

Synopsis: Package flags and behavior settings

Declaration: `Property Flags : TStrings`

Visibility: public

Access: Read

Description: `Flags` contains a set of flags that control various aspects of package behavior including installation options, build preferences, and compatibility settings.

29.40.45 TPackage.PackageVersion

Synopsis: Version information for the package

Declaration: `Property PackageVersion : TFPVersion`

Visibility: public

Access: Read

Description: `PackageVersion` contains detailed version information for the package including major, minor, micro, and build numbers. This structured version data is used for precise dependency checking.

29.40.46 TPackage.TransmitOptions

Synopsis: Options transmitted to dependent packages

Declaration: `Property TransmitOptions : TStrings`

Visibility: public

Access: Read,Write

Description: `TransmitOptions` contains compiler options that are automatically passed to packages that depend on this package. This ensures consistent compilation settings across package hierarchies.

29.40.47 TPackage.OSes

Synopsis: Supported operating systems for the package

Declaration: `Property OSes : TOSes`

Visibility: public

Access: Read,Write

Description: `OSes` specifies the set of operating systems on which this package can be built and used. This includes platforms like Windows, Linux, macOS, FreeBSD, and others supported by Free Pascal.

29.40.48 TPackage.CPUs

Synopsis: Supported CPU architectures for the package

Declaration: `Property CPUs : TCPUS`

Visibility: public

Access: Read,Write

Description: `CPUs` specifies the set of CPU architectures supported by this package. This includes architectures like x86_64, i386, ARM, PowerPC, and others supported by Free Pascal.

29.40.49 TPackage.NeedLibC

Synopsis: Whether package requires C library linkage

Declaration: `Property NeedLibC : Boolean`

Visibility: public

Access: Read,Write

Description: `NeedLibC` indicates whether this package requires linking with the C runtime library. When `True`, the appropriate C library is automatically linked during compilation.

29.40.50 TPackage.Options

Synopsis: Compiler options for the package

Declaration: `Property Options : TStrings`

Visibility: public

Access: Read,Write

Description: `Options` contains the collection of compiler command-line options that are applied when building any target within this package. These options affect all compilation within the package.

29.40.51 TPackage.UnitPath

Synopsis: Search paths for unit files

Declaration: `Property UnitPath : TConditionalStrings`

Visibility: public

Access: Read

Description: `UnitPath` contains the search paths where the compiler looks for compiled unit files (.ppu) when building targets in this package. These paths are added to the compiler's unit search directories.

29.40.52 TPackage.ObjectPath

Synopsis: Search paths for object files

Declaration: `Property ObjectPath : TConditionalStrings`

Visibility: public

Access: Read

Description: `ObjectPath` contains the search paths where the compiler and linker look for object files (.o) when building targets in this package. These paths are added to the linker's object search directories.

29.40.53 TPackage.IncludePath

Synopsis: Search paths for include files

Declaration: `Property IncludePath : TConditionalStrings`

Visibility: public

Access: Read

Description: `IncludePath` contains the search paths where the compiler looks for include files referenced by {$INCLUDE} directives. These paths are added to the compiler's include search directories.

29.40.54 TPackage.SourcePath

Synopsis: Search paths for source files

Declaration: `Property SourcePath : TConditionalStrings`

Visibility: public

Access: Read

Description: Specifies the search paths where the build system will look for Pascal source files (.pas, .pp) required for package compilation.

Multiple paths can be specified, and they are searched in order when resolving unit dependencies.

29.40.55 TPackage.ExamplePath

Synopsis: Search paths for example files

Declaration: `Property ExamplePath : TConditionalStrings`

Visibility: public

Access: Read

Description: Specifies the search paths where example programs and units are located for this package.

These paths are used when building example targets and determining installation locations for demonstration code.

29.40.56 TPackage.TestPath

Synopsis: Search paths for test files

Declaration: `Property TestPath : TConditionalStrings`

Visibility: public

Access: Read

Description: Specifies the search paths where test programs and units are located for this package.

These paths are used when building test targets and organizing the package's test suite structure.

29.40.57 TPackage.FPDocFormat

Synopsis: Output format for FPDoc documentation

Declaration: `Property FPDocFormat : TFPDocFormats`

Visibility: public

Access: Read,Write

Description: Specifies the output format for FPDoc-generated documentation (HTML, LaTeX, XML, etc.).

This property controls how the package's reference documentation is formatted when generated from source code comments and XML documentation files.

29.40.58 TPackage.InstallFiles

Synopsis: Collection of files to be installed

Declaration: `Property InstallFiles : TConditionalDestStrings`

Visibility: public

Access: Read

Description: Collection of TInstallFiles (905) objects that define which files should be installed and their target locations.

This collection manages deployment of compiled units, documentation, examples, and other package assets during installation.

29.40.59 TPackage.CleanFiles

Synopsis: Collection of files to be cleaned

Declaration: `Property CleanFiles : TConditionalStrings`

Visibility: public

Access: Read

Description: Collection of TCleanFiles (905) objects that specify which files and directories should be removed during cleanup operations.

This collection manages removal of temporary files, compiled objects, and generated outputs during clean builds.

29.40.60 TPackage.Dependencies

Synopsis: Collection of package dependencies

Declaration: `Property Dependencies : TDependencies`

Visibility: public

Access: Read

Description: Collection of TDependencies (986) objects that define other packages required for this package to build and function correctly.

Dependencies are checked during build initialization to ensure all required packages are available and properly configured.

29.40.61 TPackage.Commands

Synopsis: Collection of custom build commands

Declaration: `Property Commands : TCommands`

Visibility: public

Access: Read

Description: Collection of TCommands (960) objects that define custom build operations and external tool invocations.

Commands enable integration of external tools, pre/post-build scripts, and specialized build operations within the package build process.

29.40.62 TPackage.State

Synopsis: Current build state of the package

Declaration: `Property State : TTargetState`

Visibility: public

Access: Read,Write

Description: Indicates the current build state of the package (e.g., building, compiled, installed).

This property tracks the package's progress through the build pipeline and is used for state-dependent operations.

29.40.63 TPackage.Targets

Synopsis: Collection of build targets

Declaration: `Property Targets : TTargets`

Visibility: public

Access: Read

Description: Collection of TTargets (1061) objects that define the buildable components of this package (units, programs, libraries).

Each target represents a specific compilation unit with its own build settings, dependencies, and output configuration.

29.40.64 TPackage.Sources

Synopsis: Collection of source files

Declaration: `Property Sources : TSources`

Visibility: public

Access: Read

Description: Collection of TSources (1046) objects that manage individual source files associated with this package.

Sources include Pascal files, documentation files, examples, and other assets required for package building and distribution.

29.40.65 TPackage.UnitDir

Synopsis: Directory for compiled unit files

Declaration: `Property UnitDir : string`

Visibility: public

Access: Read,Write

Description: Specifies the directory where compiled unit files (.ppu, .o) are generated during package compilation.

This directory contains the output of successful compilation and serves as input for linking and installation operations.

29.40.66 TPackage.UnitConfigFileName

Synopsis: Filename for unit configuration file

Declaration: `Property UnitConfigFileName : string`

Visibility: public

Access: Read,Write

Description: Specifies the filename for the configuration file that contains unit search paths and compilation settings for this package.

This file is generated during build and used by the compiler to locate unit dependencies and apply package-specific settings.

29.40.67 TPackage.BeforeCompile

Synopsis: Event handler executed before package compilation

Declaration: `Property BeforeCompile : TNotifyEvent`

Visibility: public

Access: Read,Write

Description: Event handler called immediately before this package begins compilation.

This event allows custom pre-compilation setup, validation, or file preparation to be performed before the build process starts.

29.40.68 TPackage.BeforeCompileProc

Synopsis: Procedural event handler before compilation

Declaration: `Property BeforeCompileProc : TNotifyProcEvent`

Visibility: public

Access: Read,Write

Description: Procedural event handler version of BeforeCompile (1028) that provides a simpler callback interface.

This event enables procedural callback functions to be executed before package compilation begins.

29.40.69 TPackage.AfterCompile

Synopsis: Event handler executed after package compilation

Declaration: `Property AfterCompile : TNotifyEvent`

Visibility: public

Access: Read,Write

Description: Event handler called immediately after this package completes compilation successfully.

This event allows custom post-compilation processing, validation, or file operations to be performed after the build process completes.

29.40.70 TPackage.AfterCompileProc

Synopsis: Procedural event handler after compilation

Declaration: `Property AfterCompileProc : TNotifyProcEvent`

Visibility: public

Access: Read,Write

Description: Procedural event handler version of AfterCompile (1028) that provides a simpler callback interface.

This event enables procedural callback functions to be executed after package compilation completes.

29.40.71 TPackage.BeforeInstall

Synopsis: Event handler executed before package installation

Declaration: `Property BeforeInstall : TNotifyEvent`

Visibility: public

Access: Read,Write

Description: Event handler called immediately before this package begins installation.

This event allows custom pre-installation setup, directory preparation, or validation to be performed before the installation process starts.

29.40.72 TPackage.BeforeInstallProc

Synopsis: Procedural event handler before installation

Declaration: `Property BeforeInstallProc : TNotifyProcEvent`

Visibility: public

Access: Read,Write

Description: Procedural event handler version of BeforeInstall (1029) that provides a simpler callback interface.

This event enables procedural callback functions to be executed before package installation begins.

29.40.73 TPackage.AfterInstall

Synopsis: Event handler executed after package installation

Declaration: `Property AfterInstall : TNotifyEvent`

Visibility: public

Access: Read,Write

Description: Event handler called immediately after this package completes installation successfully.

This event allows custom post-installation processing, configuration, or verification to be performed after installation completes.

29.40.74 TPackage.AfterInstallProc

Synopsis: Procedural event handler after installation

Declaration: `Property AfterInstallProc : TNotifyProcEvent`

Visibility: public

Access: Read,Write

Description: Procedural event handler version of AfterInstall (1029) that provides a simpler callback interface.

This event enables procedural callback functions to be executed after package installation completes.

29.40.75 TPackage.BeforeClean

Synopsis: Event handler executed before package cleanup

Declaration: `Property BeforeClean : TNotifyEvent`

Visibility: public

Access: Read,Write

Description: Event handler called immediately before this package begins cleanup operations.

This event allows custom pre-cleanup preparation or backup operations to be performed before generated files are removed.

29.40.76 TPackage.BeforeCleanProc

Synopsis: Procedural event handler before cleanup

Declaration: `Property BeforeCleanProc : TNotifyProcEvent`

Visibility: public

Access: Read,Write

Description: Procedural event handler version of BeforeClean (1030) that provides a simpler callback interface.

This event enables procedural callback functions to be executed before package cleanup begins.

29.40.77 TPackage.AfterClean

Synopsis: Event handler executed after package cleanup

Declaration: `Property AfterClean : TNotifyEvent`

Visibility: public

Access: Read,Write

Description: Event handler called immediately after this package completes cleanup operations.

This event allows custom post-cleanup verification or additional cleanup operations to be performed after generated files are removed.

29.40.78 TPackage.AfterCleanProc

Synopsis: Procedural event handler after cleanup

Declaration: `Property AfterCleanProc : TNotifyProcEvent`

Visibility: public

Access: Read,Write

Description: Procedural event handler version of AfterClean (1030) that provides a simpler callback interface.

This event enables procedural callback functions to be executed after package cleanup completes.

29.40.79 TPackage.BeforeArchive

Synopsis: Event handler executed before package archiving

Declaration: `Property BeforeArchive : TNotifyEvent`

Visibility: public

Access: Read,Write

Description: Event handler called immediately before this package begins archive creation.

This event allows custom pre-archiving operations, file preparation, or validation to be performed before archive generation starts.

29.40.80 TPackage.BeforeArchiveProc

Synopsis: Procedural event handler before archiving

Declaration: `Property BeforeArchiveProc : TNotifyProcEvent`

Visibility: public

Access: Read,Write

Description: Procedural event handler version of BeforeArchive (1031) that provides a simpler callback interface.

This event enables procedural callback functions to be executed before package archiving begins.

29.40.81 TPackage.AfterArchive

Synopsis: Event handler executed after package archiving

Declaration: `Property AfterArchive : TNotifyEvent`

Visibility: public

Access: Read,Write

Description: Event handler called immediately after this package completes archive creation.

This event allows custom post-archiving operations, validation, or distribution to be performed after the archive is successfully generated.

29.40.82 TPackage.AfterArchiveProc

Synopsis: Procedural event handler after archiving

Declaration: `Property AfterArchiveProc : TNotifyProcEvent`

Visibility: public

Access: Read,Write

Description: Procedural event handler version of AfterArchive (1031) that provides a simpler callback interface.

This event enables procedural callback functions to be executed after package archiving completes.

29.40.83 TPackage.BeforeManifest

Synopsis: Event handler executed before manifest creation

Declaration: `Property BeforeManifest : TNotifyEvent`

Visibility: public

Access: Read,Write

Description: Event handler called immediately before this package begins manifest file generation.

This event allows custom pre-manifest operations, content preparation, or validation to be performed before manifest creation starts.

29.40.84 TPackage.BeforeManifestProc

Synopsis: Procedural event handler before manifest creation

Declaration: `Property BeforeManifestProc : TNotifyProcEvent`

Visibility: public

Access: Read,Write

Description: Procedural event handler version of BeforeManifest (1032) that provides a simpler callback interface.

This event enables procedural callback functions to be executed before manifest creation begins.

29.40.85 TPackage.AfterManifest

Synopsis: Event handler executed after manifest creation

Declaration: `Property AfterManifest : TNotifyEvent`

Visibility: public

Access: Read,Write

Description: Event handler called immediately after this package completes manifest file generation.

This event allows custom post-manifest operations, validation, or processing to be performed after the manifest is successfully created.

29.40.86 TPackage.AfterManifestProc

Synopsis: Procedural event handler after manifest creation

Declaration: `Property AfterManifestProc : TNotifyProcEvent`

Visibility: public

Access: Read,Write

Description: Procedural event handler version of AfterManifest (1032) that provides a simpler callback interface.

This event enables procedural callback functions to be executed after manifest creation completes.

29.40.87 TPackage.BeforePkgList

Synopsis: Event handler executed before package listing

Declaration: `Property BeforePkgList : TNotifyEvent`

Visibility: public

Access: Read,Write

Description: Event handler called immediately before this package generates its file listing.

This event allows custom pre-listing operations, file preparation, or validation to be performed before the package list is created.

29.40.88 TPackage.BeforePkgListProc

Synopsis: Procedural event handler before package listing

Declaration: `Property BeforePkgListProc : TNotifyProcEvent`

Visibility: public

Access: Read,Write

Description: Procedural event handler version of BeforePkgList (1033) that provides a simpler callback interface.

This event enables procedural callback functions to be executed before package listing begins.

29.40.89 TPackage.AfterPkgList

Synopsis: Event handler executed after package listing

Declaration: `Property AfterPkgList : TNotifyEvent`

Visibility: public

Access: Read,Write

Description: Event handler called immediately after this package completes its file listing generation.

This event allows custom post-listing operations, validation, or processing to be performed after the package list is created.

29.40.90 TPackage.AfterPkgListProc

Synopsis: Procedural event handler after package listing

Declaration: `Property AfterPkgListProc : TNotifyProcEvent`

Visibility: public

Access: Read,Write

Description: Procedural event handler version of AfterPkgList (1033) that provides a simpler callback interface.

This event enables procedural callback functions to be executed after package listing completes.

29.41 TPackageDictionary

29.41.1 Description

`TPackageDictionary` extends the basic dictionary functionality with package-specific variable support.

This class maintains its own variable namespace while also providing access to a master dictionary for fallback lookups.

Package dictionaries enable isolated variable scoping per package while maintaining global configuration access.

29.41.2 Method overview

Page	Method	Description
1034	GetValue	Retrieves package-specific substituted value

29.41.3 Property overview

Page	Properties	Access	Description
1034	MasterDictionary	rw	Reference to the master dictionary for fallback lookups

29.41.4 TPackageDictionary.GetValue

Synopsis: Retrieves package-specific substituted value

Declaration:
```
function GetValue(const AName: string; const Args: string) : string
                        ;  Override
```

Visibility: public

Description: `GetValue` retrieves a substituted value first from the package dictionary, then from the master dictionary if not found locally.

This method implements cascading lookup behavior for package-specific variable resolution.

The method supports both variable expansion and function callbacks with argument passing.

29.41.5 TPackageDictionary.MasterDictionary

Synopsis: Reference to the master dictionary for fallback lookups

Declaration:
```
Property MasterDictionary : TDictionary
```

Visibility: public

Access: Read,Write

Description: `MasterDictionary` provides access to the global dictionary that serves as fallback for variable lookups.

When a variable is not found in the package-specific dictionary, the lookup falls back to this master dictionary.

This property enables hierarchical variable resolution from package-specific to global scope.

29.42 TPackageEnumerator

29.42.1 Description

`TPackageEnumerator` provides iteration support for `TPackages` collections using the for-in syntax. This class implements the enumerator pattern allowing convenient iteration through package collections.

The enumerator maintains the current position and provides access to the current TPackage (1009) object during iteration loops.

29.42.2 Method overview

Page	Method	Description
1035	GetCurrent	Gets current package in enumeration

29.42.3 Property overview

Page	Properties	Access	Description
1035	Current	r	Current package being enumerated

29.42.4 TPackageEnumerator.GetCurrent

Synopsis: Gets current package in enumeration

Declaration: `function GetCurrent : TPackage`

Visibility: public

Description: `GetCurrent` returns the TPackage (1009) object at the current position in the enumeration. This method is used internally by the enumerator to provide the current package during for-in loops.

29.42.5 TPackageEnumerator.Current

Synopsis: Current package being enumerated

Declaration: `Property Current : TPackage`

Visibility: public

Access: Read

Description: `Current` property provides read-only access to the TPackage (1009) object at the current position during enumeration. This property is used by for-in loops to access the current package.

29.43 TPackages

29.43.1 Description

`TPackages` is a managed collection class that stores and provides access to multiple TPackage (1009) objects. This class supports enumeration, indexed access, and package lookup by name.

The collection manages package objects and provides various methods for adding, accessing, and iterating through packages in the fpmake build system.

29.43.2 Method overview

Page	Method	Description
1036	AddPackage	Adds a new package to the collection
1036	GetEnumerator	Gets enumerator for iterating packages

29.43.3 Property overview

Page	Properties	Access	Description
1037	PackageItems	rw	Indexed access to packages in the collection
1036	Packages	r	Access to packages by name

29.43.4 TPackages.AddPackage

Synopsis: Adds a new package to the collection

Declaration: `function AddPackage(const AName: string) : TPackage`

Visibility: public

Description: `AddPackage` creates and adds a new TPackage (1009) object to the collection with the specified name. The method returns the newly created package object for further configuration.

Each package in the collection must have a unique name. The created package can then be configured with targets, dependencies, and other package-specific settings.

29.43.5 TPackages.GetEnumerator

Synopsis: Gets enumerator for iterating packages

Declaration: `function GetEnumerator : TPackageEnumerator`

Visibility: public

Description: `GetEnumerator` returns a TPackageEnumerator (1035) object that enables for-in loop iteration through all packages in the collection.

This method is called automatically by the compiler when using for-in syntax with package collections.

29.43.6 TPackages.Packages

Synopsis: Access to packages by name

Declaration: `Property Packages[AName: string]: TPackage; default`

Visibility: public

Access: Read

Description: `Packages` property provides name-based access to TPackage (1009) objects in the collection. This allows retrieving packages by their string name identifier.

If a package with the specified name does not exist, the property returns `nil`.

29.43.7 TPackages.PackageItems

Synopsis: Indexed access to packages in the collection

Declaration: `Property PackageItems[AIndex: Integer]: TPackage`

Visibility: public

Access: Read,Write

Description: `PackageItems` provides zero-based indexed access to TPackage (1009) objects in the collection. The index must be within the range 0 to `Count`-1.

This property enables accessing packages by their position in the collection rather than by name.

29.44 TPackageVariant

29.44.1 Description

`TPackageVariant` represents a specific configuration variant of a package with custom build options and settings. This class allows packages to define multiple build configurations with different compiler options, target platforms, or conditional compilation settings.

29.44.2 Method overview

Page	Method	Description
1037	Create	Creates a new package variant configuration
1037	Destroy	Destroys package variant and frees resources

29.44.3 Property overview

Page	Properties	Access	Description
1038	IncludePath	r	Include file search paths for this variant
1038	Options	r	Compiler options for this variant
1038	SourcePath	r	Source file search paths for this variant
1038	Targets	r	Build targets associated with this variant

29.44.4 TPackageVariant.Create

Synopsis: Creates a new package variant configuration

Declaration: `constructor Create(ACollection: TCollection);  Override`

Visibility: public

Description: Creates a new `TPackageVariant` instance and adds it to the specified collection. The variant is initialized with default settings and ready for configuration with specific build options.

29.44.5 TPackageVariant.Destroy

Synopsis: Destroys package variant and frees resources

Declaration: `destructor Destroy;  Override`

Visibility: public

Description: Destroys the `TPackageVariant` instance and releases all associated resources including option collections and configuration data. This method is called automatically when the variant is no longer needed.

29.44.6 TPackageVariant.Options

Synopsis: Compiler options for this variant

Declaration: `Property Options : TStrings`

Visibility: public

Access: Read

Description: `Options` contains the collection of compiler command-line options specific to this package variant. These options are passed to the compiler when building the package with this variant configuration.

29.44.7 TPackageVariant.IncludePath

Synopsis: Include file search paths for this variant

Declaration: `Property IncludePath : TConditionalStrings`

Visibility: public

Access: Read

Description: `IncludePath` specifies the search paths for include files when compiling with this variant. These paths are added to the compiler's include search directories.

29.44.8 TPackageVariant.SourcePath

Synopsis: Source file search paths for this variant

Declaration: `Property SourcePath : TConditionalStrings`

Visibility: public

Access: Read

Description: `SourcePath` specifies the search paths for source unit files when compiling with this variant. These paths are added to the compiler's unit search directories.

29.44.9 TPackageVariant.Targets

Synopsis: Build targets associated with this variant

Declaration: `Property Targets : TTargets`

Visibility: public

Access: Read

Description: `Targets` contains the collection of build targets that are built when using this package variant. Each target in this collection will be compiled with the variant's specific options and settings.

29.45 TPackageVariantEnumerator

29.45.1 Description

`TPackageVariantEnumerator` provides enumeration support for iterating through collections of package variants using for-in loop syntax. This enumerator implements the standard enumeration pattern for Pascal collections.

29.45.2 Method overview

Page	Method	Description
1039	GetCurrent	Gets current package variant in enumeration

29.45.3 Property overview

Page	Properties	Access	Description
1039	Current	r	Current package variant being enumerated

29.45.4 TPackageVariantEnumerator.GetCurrent

Synopsis: Gets current package variant in enumeration

Declaration: `function GetCurrent : TPackageVariant`

Visibility: public

Description: Returns the current `TPackageVariant` object in the enumeration sequence. This method is called internally by the for-in loop mechanism to access the current variant.

29.45.5 TPackageVariantEnumerator.Current

Synopsis: Current package variant being enumerated

Declaration: `Property Current : TPackageVariant`

Visibility: public

Access: Read

Description: `Current` property provides read-only access to the current `TPackageVariant` in the enumeration. This property is used by for-in loops to access the variant object during iteration.

29.46 TPackageVariants

29.46.1 Description

`TPackageVariants` is a collection class that manages multiple `TPackageVariant` objects. This class provides methods for adding, accessing, and enumerating package variants within a package configuration.

29.46.2 Method overview

Page	Method	Description
1040	Add	Adds a new package variant to the collection
1040	GetEnumerator	Gets enumerator for iterating package variants

29.46.3 Property overview

Page	Properties	Access	Description
1041	ActivePackageVariant	r	Currently active package variant for building
1042	ActivePackageVariantName	rw	Name of the currently active package variant
1042	AutoAddToPackage	r	Whether variants are automatically added to the package
1041	DefaultPackageVariant	r	Default package variant used when none specified
1041	DefaultPackageVariantName	rw	Name of the default package variant
1042	IsInheritable	r	Whether package variants can be inherited by child packages
1041	MasterPackage	r	Reference to the master package owning the variants
1040	Name	rw	Name of the package variants collection

29.46.4 TPackageVariants.Add

Synopsis: Adds a new package variant to the collection

Declaration: `function Add(AName: string) : TPackageVariant; Virtual; Overload`

Visibility: public

Description: Creates and adds a new `TPackageVariant` to the collection. The new variant is initialized with default settings and can then be configured with specific build options and targets.

29.46.5 TPackageVariants.GetEnumerator

Synopsis: Gets enumerator for iterating package variants

Declaration: `function GetEnumerator : TPackageVariantEnumerator`

Visibility: public

Description: Returns a `TPackageVariantEnumerator` that can be used to iterate through all package variants in the collection using for-in loop syntax.

29.46.6 TPackageVariants.Name

Synopsis: Name of the package variants collection

Declaration: `Property Name : string`

Visibility: public

Access: Read,Write

Description: `Name` property contains the identifier for this package variants collection. This name is used to reference the collection within the package configuration.

29.46.7 TPackageVariants.MasterPackage

Synopsis: Reference to the master package owning the variants

Declaration: `Property MasterPackage : TPackage`

Visibility: public

Access: Read

Description: `MasterPackage` provides access to the `TPackage` instance that owns this variants collection. This reference allows variants to access package-level settings and configuration.

29.46.8 TPackageVariants.DefaultPackageVariant

Synopsis: Default package variant used when none specified

Declaration: `Property DefaultPackageVariant : TPackageVariant`

Visibility: public

Access: Read

Description: `DefaultPackageVariant` specifies the package variant to use when no specific variant is requested during the build process. This ensures consistent build behavior when variants are not explicitly selected.

29.46.9 TPackageVariants.ActivePackageVariant

Synopsis: Currently active package variant for building

Declaration: `Property ActivePackageVariant : TPackageVariant`

Visibility: public

Access: Read

Description: `ActivePackageVariant` represents the package variant currently selected for building. This property determines which variant's settings and options are applied during the build process.

29.46.10 TPackageVariants.DefaultPackageVariantName

Synopsis: Name of the default package variant

Declaration: `Property DefaultPackageVariantName : string`

Visibility: public

Access: Read,Write

Description: `DefaultPackageVariantName` contains the string name of the default package variant. This name is used to identify which variant should be used when no specific variant is selected.

29.46.11 TPackageVariants.ActivePackageVariantName

Synopsis: Name of the currently active package variant

Declaration: `Property ActivePackageVariantName : string`

Visibility: public

Access: Read,Write

Description: `ActivePackageVariantName` contains the string name of the currently active package variant. This property reflects which variant is selected for the current build operation.

29.46.12 TPackageVariants.IsInheritable

Synopsis: Whether package variants can be inherited by child packages

Declaration: `Property IsInheritable : Boolean`

Visibility: public

Access: Read

Description: `IsInheritable` determines whether the package variants in this collection can be inherited by dependent child packages, allowing for consistent variant propagation through package hierarchies.

29.46.13 TPackageVariants.AutoAddToPackage

Synopsis: Whether variants are automatically added to the package

Declaration: `Property AutoAddToPackage : Boolean`

Visibility: public

Access: Read

Description: `AutoAddToPackage` controls whether new variants created in this collection are automatically registered with the parent package. When `True`, variants are immediately available for building.

29.47 TResourceFile

29.47.1 Description

`TResourceFile` represents a resource file that needs to be processed and included during package compilation.

Resource files contain binary data, icons, bitmaps, strings, or other assets that are embedded into the compiled executable.

The build system processes resource files according to platform-specific resource compilation requirements.

29.48 TResourceFiles

29.48.1 Description

`TResourceFiles` is a specialized collection class that manages resource files included in Free Pascal package builds.

This collection handles resource file tracking, installation paths, and deployment configuration for binary resources, documentation files, and other assets.

29.48.2 Method overview

Page	Method	Description
1043	GetInstallFiles	Gets list of resource files for installation

29.48.3 TResourceFiles.GetInstallFiles

Synopsis: Gets list of resource files for installation

Declaration:
```
procedure GetInstallFiles(AList: TStrings; const APrefixU: string;
                          const APrefixB: string; ACPU: TCpu; AOS: TOS)
                          ;  Virtual
```

Visibility: public

Description: Populates the provided list with resource files that should be installed as part of this resource collection's deployment process.

This method collects all resources associated with this collection that require installation or distribution with the package.

29.49 TSource

29.49.1 Description

`TSource` represents a single source code file within the Free Pascal build system, tracking file paths, names, and build-related properties.

This class manages individual source files that are compiled as part of build targets, including dependency tracking and compilation settings.

29.49.2 Method overview

Page	Method	Description
1044	Create	Creates a new source file entry
1044	Destroy	Destroys the source file entry and releases resources
1044	GetInstallFiles	Gets list of files for installation from this source

29.49.3 Property overview

Page	Properties	Access	Description
1045	InstallSourcePath	r	Installation path for the source file
1044	SourceType	r	Type of source file (documentation, example, etc.)

29.49.4 TSource.Create

Synopsis: Creates a new source file entry

Declaration: `constructor Create(ACollection: TCollection);  Override`

Visibility: public

Description: Initializes a new `TSource` instance within the specified collection, setting up the source file entry for build processing.

The source entry is configured with default settings and linked to its parent collection for management during the build process.

29.49.5 TSource.Destroy

Synopsis: Destroys the source file entry and releases resources

Declaration: `destructor Destroy;  Override`

Visibility: public

Description: Releases all resources associated with the source file entry and performs cleanup of any allocated memory or references.

This destructor ensures proper cleanup when the source file entry is removed from the build system.

29.49.6 TSource.GetInstallFiles

Synopsis: Gets list of files for installation from this source

Declaration: `procedure GetInstallFiles(List: TStrings);  Virtual`

Visibility: public

Description: Populates the provided list with files that should be installed as part of this source file's deployment process.

This method collects all files associated with this source entry that require installation or distribution with the package.

29.49.7 TSource.SourceType

Synopsis: Type of source file (documentation, example, etc.)

Declaration: `Property SourceType : TSourceType`

Visibility: public

Access: Read

Description: Specifies the category or purpose of this source file within the build system.

The source type determines how the file is processed, where it is installed, and what build rules apply to it.

29.49.8 TSource.InstallSourcePath

Synopsis: Installation path for the source file

Declaration: `Property InstallSourcePath : string`

Visibility: public

Access: Read

Description: Specifies the target directory path where this source file should be installed during package deployment.

This path is relative to the installation base directory and determines the final location of the source file after installation.

29.50 TSourceEnumerator

29.50.1 Description

`TSourceEnumerator` provides enumeration support for iterating through collections of source files using for-in loop constructs.

This enumerator enables convenient iteration over source file collections with automatic position tracking and bounds checking.

29.50.2 Method overview

Page	Method	Description
1045	GetCurrent	Gets current source file in enumeration

29.50.3 Property overview

Page	Properties	Access	Description
1045	Current	r	Current source file being enumerated

29.50.4 TSourceEnumerator.GetCurrent

Synopsis: Gets current source file in enumeration

Declaration: `function GetCurrent : TSource`

Visibility: public

Description: Returns the TSource (1043) object at the current enumeration position.

This method provides access to the source file's properties and methods during iteration through the source collection.

29.50.5 TSourceEnumerator.Current

Synopsis: Current source file being enumerated

Declaration: `Property Current : TSource`

Visibility: public

Access: Read

Description: Property providing read-only access to the TSource (1043) object at the current enumeration position.

This property returns the source file object that can be processed during for-in loop iteration.

29.51 TSources

29.51.1 Description

`TSources` is a collection class that manages multiple source file entries for a Free Pascal package.

This collection provides methods for adding different types of sources (documentation, examples, tests) and manages their properties for building and installation.

29.51.2 Method overview

Page	Method	Description
1046	AddDoc	Adds documentation sources to the collection
1048	AddDocFiles	Adds documentation files matching file mask pattern
1047	AddExample	Adds example sources to the collection
1048	AddExampleFiles	Adds example files matching file mask pattern
1046	AddSrc	Adds source files to the collection
1048	AddSrcFiles	Adds source files matching file mask pattern
1047	AddTest	Adds test sources to the collection
1048	AddTestFiles	Adds test files to the source collection
1047	GetEnumerator	Gets enumerator for iterating sources

29.51.3 Property overview

Page	Properties	Access	Description
1049	SourceItems	rw	Indexed access to source files in the collection

29.51.4 TSources.AddDoc

Synopsis: Adds documentation sources to the collection

Declaration:
```
function AddDoc(const AFiles: string) : TSource
function AddDoc(const AFiles: string; AInstallSourcePath: string)
                : TSource
```

Visibility: public

Description: Creates and adds a new documentation source entry that specifies files to be installed in the documentation directory. The files are copied during installation to the configured documentation installation path.

Documentation sources typically include readme files, manuals, API documentation, and other informational content for the package.

29.51.5 TSources.AddSrc

Synopsis: Adds source files to the collection

Declaration: `function AddSrc(const AFiles: string) : TSource`

Visibility: public

Description: Creates and adds a new source entry that specifies Pascal source files (.pp, .pas) to be distributed with the package. These files are typically installed for developers who want to examine or modify the package source code.

Source files are separate from compiled units and provide transparency and modification capabilities for package users.

29.51.6 TSources.AddExample

Synopsis: Adds example sources to the collection

Declaration:
```
function AddExample(const AFiles: string) : TSource
function AddExample(const AFiles: string; AInstallSourcePath: string)
                    : TSource
```

Visibility: public

Description: Creates and adds a new example source entry that specifies example program files to be installed in the examples directory. Examples demonstrate usage of the package functionality and serve as learning resources.

Example files help users understand how to properly use the package APIs and provide working code samples for common use cases.

29.51.7 TSources.AddTest

Synopsis: Adds test sources to the collection

Declaration: `function AddTest(const AFiles: string) : TSource`

Visibility: public

Description: Creates and adds a new test source entry that specifies test program files used for package validation and quality assurance. Test files verify that the package functionality works correctly.

Test sources are typically run during development and before package releases to ensure code quality and prevent regressions.

29.51.8 TSources.GetEnumerator

Synopsis: Gets enumerator for iterating sources

Declaration: `function GetEnumerator : TSourceEnumerator`

Visibility: public

Description: Returns a TSourceEnumerator (1045) object that enables for-in loop iteration through the source files collection.

This method enables convenient enumeration of all source files using modern Pascal iteration syntax.

29.51.9 TSources.AddDocFiles

Synopsis: Adds documentation files matching file mask pattern

Declaration:
```
procedure AddDocFiles(const AFileMask: string;
                      const ASearchPathPrefix: string;
                      Recursive: Boolean; AInstallSourcePath: string)
```

Visibility: public

Description: Searches for and adds documentation files that match the specified file mask pattern to the source collection.

This method provides a convenient way to add multiple documentation files at once using wildcard patterns for file selection.

29.51.10 TSources.AddSrcFiles

Synopsis: Adds source files matching file mask pattern

Declaration:
```
procedure AddSrcFiles(const AFileMask: string;
                      const ASearchPathPrefix: string;
                      Recursive: Boolean)
```

Visibility: public

Description: Searches for and adds source files that match the specified file mask pattern to the source collection.

This method provides a convenient way to add multiple source files at once using wildcard patterns for file selection.

29.51.11 TSources.AddExampleFiles

Synopsis: Adds example files matching file mask pattern

Declaration:
```
procedure AddExampleFiles(const AFileMask: string;
                          const ASearchPathPrefix: string;
                          Recursive: Boolean; AInstallSourcePath: string)
```

Visibility: public

Description: `AddExampleFiles` adds example files to the source collection that match the specified file mask pattern. The method scans for files matching `AFileMask` and adds them to be included with the package as example code.

Example files are typically demonstration code showing how to use the package functionality. They are installed to the specified installation path for reference by package users.

29.51.12 TSources.AddTestFiles

Synopsis: Adds test files to the source collection

Declaration:
```
procedure AddTestFiles(const AFileMask: string;
                       const ASearchPathPrefix: string;
                       Recursive: Boolean)
```

Visibility: public

Description: `AddTestFiles` adds test files matching the specified file mask pattern to the source collection. Test files contain unit tests and validation code for verifying package functionality.

The method searches for files matching `AFileMask` and includes them in the package for testing purposes. Test files are used during package validation and quality assurance.

29.51.13 TSources.SourceItems

Synopsis: Indexed access to source files in the collection

Declaration: `Property SourceItems[Index: Integer]: TSource; default`

Visibility: public

Access: Read,Write

Description: `SourceItems` provides indexed access to individual source files in the `TSources` collection. This property allows retrieving TSourceItem (905) objects by their numerical index.

The index is zero-based and must be within the range 0 to `Count`-1. Each source item contains information about a single source file including its name, path, and installation settings.

29.52 TTarget

29.52.1 Description

`TTarget` represents a single build target within a package, such as a unit, program, or library. Each target defines what needs to be compiled, how it should be compiled, and where the resulting files should be installed.

Build targets can be units, programs, libraries, or documentation files. They contain information about source files, compilation options, dependencies, CPU and OS requirements, and installation settings.

Targets are managed through the TTargets (1061) collection and are owned by TPackage (1009) instances.

See also: TPackage (1009), TTargets (1061), TTargetType (930), TFileType (921), TDependency (988)

29.52.2 Method overview

Page	Method	Description
1051	AssignTo	Assigns target properties to another object
1050	Create	Creates a new build target instance
1050	Destroy	Destroys the target instance and frees resources
1052	GetArchiveFiles	Gets list of files to be archived for this target
1052	GetCleanFiles	Gets list of files to be cleaned for this target
1052	GetInstallFiles	Gets list of files to be installed for this target
1051	GetOutputFileName	Returns the target's output filename for the specified OS
1051	HaveOptions	Checks if target has compilation options
1051	SetExeName	Sets the executable name for the target
1051	SetName	Sets the name of the target
1052	SetXML	Sets XML configuration data for the target

29.52.3 Property overview

Page	Properties	Access	Description
1060	AfterClean	rw	Event handler executed after target cleanup
1059	AfterCompile	rw	Event handler executed after target compilation
1060	BeforeClean	rw	Event handler executed before target cleanup
1059	BeforeCompile	rw	Event handler executed before target compilation
1053	Commands	r	Collection of custom commands for the target
1054	CPUs	rw	Set of CPU architectures this target supports
1053	Dependencies	r	Collection of target dependencies
1057	Directory	rw	Directory where the target source files are located
1056	Extension	rw	File extension for the compiled target
1057	FileType	rw	Type of file this target generates
1056	FPCTarget	rw	FPC compiler target specification
1058	IncludePath	r	Search path for include files during compilation
1057	Install	rw	Whether this target should be installed
1059	IsFPMakePlugin	rw	Whether this target is an FPMake plugin
1054	Mode	rw	Compiler mode for the target
1055	ObjectFileName	r	Resolved object file name for the target
1058	ObjectPath	r	Path for object files generated during compilation
1055	Options	rw	Compiler options for the target
1054	OSes	rw	Set of operating systems this target supports
1053	ResourceFiles	r	Collection of resource files for the target
1057	ResourceStrings	rw	Whether to generate resource string support for the target
1056	RSJFileName	r	Resource string JSON filename for the target
1056	RSTFileName	r	Resource string table filename for the target
1055	SourceFileName	r	Resolved source file name for the target
1053	State	r	Current build state of the target
1058	TargetSourceFileName	r	Source filename for the build target
1054	TargetType	rw	Type of the build target
1055	UnitFileName	r	Resolved unit file name for the target
1058	UnitPath	r	Search path for unit files during compilation
1059	XML	rw	Whether to generate XML documentation for the target

29.52.4 TTarget.Create

Synopsis: Creates a new build target instance

Declaration: `constructor Create(ACollection: TCollection);  Override`

Visibility: public

Description: Creates a new `TTarget` instance within the specified collection.

The target is initialized with default settings and can be configured for compilation, installation, and dependency management.

29.52.5 TTarget.Destroy

Synopsis: Destroys the target instance and frees resources

Declaration: `destructor Destroy;  Override`

Visibility: public

Description: Destroys the `TTarget` instance and frees all associated resources including dependencies, resource files, commands, and path collections.

29.52.6 TTarget.AssignTo

Synopsis: Assigns target properties to another object

Declaration: `procedure AssignTo(Dest: TPersistent);  Override`

Visibility: public

Description: Assigns the properties of this target to another persistent object.

This method is used for copying target configuration during build operations.

29.52.7 TTarget.GetOutputFileName

Synopsis: Returns the target's output filename for the specified OS

Declaration: `function GetOutputFileName(AOs: TOS) : string;  Virtual`

Visibility: public

Description: Returns the complete output filename that will be generated for this target on the specified operating system.

The filename includes appropriate extensions and paths based on the target type and OS conventions.

29.52.8 TTarget.HaveOptions

Synopsis: Checks if target has compilation options

Declaration: `function HaveOptions : Boolean`

Visibility: public

Description: Returns `True` if the target has compiler options defined, `False` otherwise.

29.52.9 TTarget.SetName

Synopsis: Sets the name of the target

Declaration: `procedure SetName(const AValue: string);  Override`

Visibility: public

Description: Sets the name of the target, which is used for identification and display purposes.

The name must be unique within the containing package.

29.52.10 TTarget.SetExeName

Synopsis: Sets the executable name for the target

Declaration: `procedure SetExeName(const AValue: string)`

Visibility: public

Description: Sets the executable filename that will be generated for program targets.

This overrides the default naming based on the target name.

29.52.11 TTarget.SetXML

Synopsis: Sets XML configuration data for the target

Declaration:
```
procedure SetXML(const AValue: string)
```

Visibility: public

Description: Sets XML configuration data associated with the target.

This XML data can contain target-specific build configuration and metadata.

29.52.12 TTarget.GetCleanFiles

Synopsis: Gets list of files to be cleaned for this target

Declaration:
```
procedure GetCleanFiles(List: TStrings; const APrefixU: string;
                        const APrefixB: string; ACPU: TCpu; AOS: TOS)
                        ;  Virtual
```

Visibility: public

Description: Populates the provided string list with filenames that should be deleted during clean operations for this target. This includes compiled object files, executables, and other generated artifacts.

The file paths are adjusted according to the CPU, OS, and subtarget parameters to match the actual build output locations.

29.52.13 TTarget.GetInstallFiles

Synopsis: Gets list of files to be installed for this target

Declaration:
```
procedure GetInstallFiles(List: TStrings; const APrefixU: string;
                          const APrefixB: string; ACPU: TCpu; AOS: TOS)
                          ;  Virtual
```

Visibility: public

Description: Populates the provided string list with filenames that should be copied during installation operations for this target. This includes compiled units, binaries, documentation, and resource files.

Installation paths are calculated based on the target type, CPU, OS, and the configured installation directories for units, binaries, and documentation.

29.52.14 TTarget.GetArchiveFiles

Synopsis: Gets list of files to be archived for this target

Declaration:
```
procedure GetArchiveFiles(List: TStrings; ACPU: TCpu; AOS: TOS)
                          ;  Virtual
```

Visibility: public

Description: Populates the provided string list with filenames that should be included in distribution archives for this target. This typically includes compiled binaries, documentation, and required resource files.

Files are filtered based on the target CPU and OS to ensure only relevant files are included in platform-specific archives.

29.52.15 TTarget.Dependencies

Synopsis: Collection of target dependencies

Declaration: `Property Dependencies : TDependencies`

Visibility: public

Access: Read

Description: Collection containing all dependencies required by this target.

Dependencies include units, packages, and include files that must be available for successful compilation of this target.

29.52.16 TTarget.ResourceFiles

Synopsis: Collection of resource files for the target

Declaration: `Property ResourceFiles : TResourceFiles`

Visibility: public

Access: Read

Description: Collection containing resource files that will be linked into this target.

Resource files can include icons, bitmaps, version information, and other embedded resources.

29.52.17 TTarget.Commands

Synopsis: Collection of custom commands for the target

Declaration: `Property Commands : TCommands`

Visibility: public

Access: Read

Description: Collection containing custom commands that will be executed during build phases.

Commands can be configured to run before or after compilation, installation, cleaning, or archiving.

29.52.18 TTarget.State

Synopsis: Current build state of the target

Declaration: `Property State : TTargetState`

Visibility: public

Access: Read

Description: Current state of the target in the build process.

States include neutral, considering, no compile, compiled, installed, and not found.

29.52.19 TTarget.TargetType

Synopsis: Type of the build target

Declaration: `Property TargetType : TTargetType`

Visibility: public

Access: Read,Write

Description: Specifies the type of target: program, unit, implicit unit, clean-only unit, example unit, example program, fpdoc documentation, or shared library.

The target type determines compilation options and output file handling.

29.52.20 TTarget.OSes

Synopsis: Set of operating systems this target supports

Declaration: `Property OSes : TOSes`

Visibility: public

Access: Read,Write

Description: Set of operating systems for which this target should be compiled.

If empty, the target is compiled for all operating systems. Used for conditional compilation.

29.52.21 TTarget.CPUs

Synopsis: Set of CPU architectures this target supports

Declaration: `Property CPUs : TCPUS`

Visibility: public

Access: Read,Write

Description: Set of CPU architectures for which this target should be compiled.

If empty, the target is compiled for all CPU architectures. Used for conditional compilation.

29.52.22 TTarget.Mode

Synopsis: Compiler mode for the target

Declaration: `Property Mode : TCompilerMode`

Visibility: public

Access: Read,Write

Description: Compiler mode used when compiling this target.

Modes include FPC, Turbo Pascal, Object Pascal, Delphi, Mac Pascal, and Delphi Unicode.

29.52.23 TTarget.Options

Synopsis: Compiler options for the target

Declaration: `Property Options : TStrings`

Visibility: public

Access: Read,Write

Description: String list containing compiler options that will be passed to the Free Pascal Compiler when compiling this target.

Options include optimization settings, debugging flags, and compiler directives.

29.52.24 TTarget.SourceFileName

Synopsis: Resolved source file name for the target

Declaration: `Property SourceFileName : string`

Visibility: public

Access: Read

Description: Full path to the main source file for this target after path resolution.

This is the primary Pascal source file that will be compiled for this target.

29.52.25 TTarget.UnitFileName

Synopsis: Resolved unit file name for the target

Declaration: `Property UnitFileName : string`

Visibility: public

Access: Read

Description: Full path to the compiled unit file (.ppu) that will be generated for this target.

This file contains the compiled unit interface and implementation.

29.52.26 TTarget.ObjectFileName

Synopsis: Resolved object file name for the target

Declaration: `Property ObjectFileName : string`

Visibility: public

Access: Read

Description: Full path to the object file (.o) that will be generated for this target.

This file contains the compiled machine code for the target.

29.52.27 TTarget.RSTFileName

Synopsis: Resource string table filename for the target

Declaration: `Property RSTFileName : string`

Visibility: public

Access: Read

Description: `RSTFileName` contains the full path to the resource string table file (.rst) for this target. This file contains string resources and translation information used by the application.

29.52.28 TTarget.RSJFileName

Synopsis: Resource string JSON filename for the target

Declaration: `Property RSJFileName : string`

Visibility: public

Access: Read

Description: `RSJFileName` contains the full path to the resource string JSON file (.rsj) for this target. This file contains string resources in JSON format for internationalization support.

29.52.29 TTarget.FPCTarget

Synopsis: FPC compiler target specification

Declaration: `Property FPCTarget : string`

Visibility: public

Access: Read,Write

Description: `FPCTarget` specifies the target platform and architecture for the Free Pascal Compiler. This setting determines the output format and platform-specific code generation for the target.

29.52.30 TTarget.Extension

Synopsis: File extension for the compiled target

Declaration: `Property Extension : string`

Visibility: public

Access: Read,Write

Description: `Extension` contains the file extension (without the dot) for the compiled output file. This extension varies based on the target type, such as 'exe' for programs or 'dll' for libraries.

29.52.31 TTarget.FileType

Synopsis: Type of file this target generates

Declaration: `Property FileType : TFileType`

Visibility: public

Access: Read,Write

Description: `FileType` indicates the category of file this target produces, such as executable program, dynamic library, static library, or unit. This determines compilation and linking behavior.

29.52.32 TTarget.Directory

Synopsis: Directory where the target source files are located

Declaration: `Property Directory : string`

Visibility: public

Access: Read,Write

Description: `Directory` specifies the directory path containing the source files for this target. This path is used to resolve relative file references during compilation.

29.52.33 TTarget.ResourceStrings

Synopsis: Whether to generate resource string support for the target

Declaration: `Property ResourceStrings : Boolean`

Visibility: public

Access: Read,Write

Description: `ResourceStrings` determines whether the compiler generates resource string support files for this target. When `True`, .rst and .rsj files are created for internationalization.

29.52.34 TTarget.Install

Synopsis: Whether this target should be installed

Declaration: `Property Install : Boolean`

Visibility: public

Access: Read,Write

Description: `Install` controls whether this target is included in the installation process. When `True`, the compiled target file is copied to the installation directory.

29.52.35 TTarget.TargetSourceFileName

Synopsis: Source filename for the build target

Declaration: `Property TargetSourceFileName : string`

Visibility: public

Access: Read

Description: `TargetSourceFileName` contains the name of the main source file for this target. This is the primary Pascal source file that serves as the entry point for compilation.

29.52.36 TTarget.ObjectPath

Synopsis: Path for object files generated during compilation

Declaration: `Property ObjectPath : TConditionalStrings`

Visibility: public

Access: Read

Description: `ObjectPath` specifies the directory path where object files (.o) are stored during compilation. This allows separation of intermediate compilation files from source code directories.

29.52.37 TTarget.UnitPath

Synopsis: Search path for unit files during compilation

Declaration: `Property UnitPath : TConditionalStrings`

Visibility: public

Access: Read

Description: `UnitPath` contains the search directories for compiled unit files (.ppu) during compilation. The compiler searches these paths to locate required units referenced by this target.

29.52.38 TTarget.IncludePath

Synopsis: Search path for include files during compilation

Declaration: `Property IncludePath : TConditionalStrings`

Visibility: public

Access: Read

Description: `IncludePath` contains the search directories for include files during compilation. The compiler searches these paths to locate files referenced by {$INCLUDE} directives.

29.52.39 TTarget.XML

Synopsis: Whether to generate XML documentation for the target

Declaration: `Property XML : string`

Visibility: public

Access: Read,Write

Description: `XML` determines whether the compiler generates XML documentation output for this target. When `True`, structured documentation is extracted from source code comments.

29.52.40 TTarget.IsFPMakePlugin

Synopsis: Whether this target is an FPMake plugin

Declaration: `Property IsFPMakePlugin : Boolean`

Visibility: public

Access: Read,Write

Description: `IsFPMakePlugin` indicates whether this target is a plugin for the FPMake build system. Plugin targets provide extended functionality and are loaded dynamically during the build process.

29.52.41 TTarget.BeforeCompile

Synopsis: Event handler executed before target compilation

Declaration: `Property BeforeCompile : TNotifyEvent`

Visibility: public

Access: Read,Write

Description: `BeforeCompile` is an event handler called immediately before this target is compiled. This allows custom preprocessing or validation logic to be executed before compilation begins.

29.52.42 TTarget.AfterCompile

Synopsis: Event handler executed after target compilation

Declaration: `Property AfterCompile : TNotifyEvent`

Visibility: public

Access: Read,Write

Description: `AfterCompile` is an event handler called immediately after this target is successfully compiled. This allows custom post-processing or validation logic to be executed after compilation completes.

29.52.43 TTarget.BeforeClean

Synopsis: Event handler executed before target cleanup

Declaration: `Property BeforeClean : TNotifyEvent`

Visibility: public

Access: Read,Write

Description: `BeforeClean` is an event handler called before this target's generated files are cleaned. This allows custom cleanup preparation or backup operations before file removal.

29.52.44 TTarget.AfterClean

Synopsis: Event handler executed after target cleanup

Declaration: `Property AfterClean : TNotifyEvent`

Visibility: public

Access: Read,Write

Description: `AfterClean` is an event handler called immediately after this target's generated files have been cleaned and removed.

This allows custom post-cleanup operations, logging, or verification that cleanup completed successfully.

29.53 TTargetEnumerator

29.53.1 Description

`TTargetEnumerator` provides enumeration support for iterating through collections of build targets in a for-in loop construct.

This enumerator enables convenient iteration over target collections with automatic position tracking and bounds checking.

29.53.2 Method overview

Page	Method	Description
1060	GetCurrent	Gets current build target in enumeration

29.53.3 Property overview

Page	Properties	Access	Description
1061	Current	r	Current build target being enumerated

29.53.4 TTargetEnumerator.GetCurrent

Synopsis: Gets current build target in enumeration

Declaration: `function GetCurrent : TTarget`

Visibility: public

Description: Returns the TTarget (1049) object at the current enumeration position.

This method provides access to the target's properties and methods during iteration through the target collection.

29.53.5 TTargetEnumerator.Current

Synopsis: Current build target being enumerated

Declaration: `Property Current : TTarget`

Visibility: public

Access: Read

Description: Property providing read-only access to the TTarget (1049) object at the current enumeration position.

This property returns the target object that can be processed during for-in loop iteration.

29.54 TTargets

29.54.1 Description

`TTargets` is a collection class that manages multiple TTarget (1049) instances within a package. It provides methods to add different types of targets such as units, programs, libraries, and documentation targets.

The collection supports various convenience methods for adding specific target types with optional CPU and OS restrictions, making it easy to create cross-platform build configurations.

See also: TTarget (1049), TTargetEnumerator (1060), TPackage (1009)

29.54.2 Method overview

Page	Method	Description
1063	AddExampleProgram	Adds an example program target to the collection
1063	AddExampleUnit	Adds an example unit target to the collection
1061	AddFPDoc	Adds a documentation target to the collection
1062	AddImplicitUnit	Adds an implicit unit target to the collection
1063	AddLibrary	Adds a library target to the collection
1062	AddProgram	Adds a program target to the collection
1062	AddUnit	Adds a unit target to the collection
1064	GetEnumerator	Gets enumerator for iterating build targets

29.54.3 Property overview

Page	Properties	Access	Description
1064	TargetItems	rw	Indexed access to build targets in the collection
1064	Targets	r	Access to build target by name

29.54.4 TTargets.AddFPDoc

Synopsis: Adds a documentation target to the collection

Declaration:
```
function AddFPDoc(const AUnitName: string; const AXMLName: string)
                  : TTarget
```

Visibility: public

Description: Creates and adds a new FPDoc documentation target that generates HTML or other documentation formats from Pascal source code and XML documentation files.

The documentation target processes the specified unit file and corresponding XML documentation file to produce formatted documentation output.

29.54.5 TTargets.AddUnit

Synopsis: Adds a unit target to the collection

Declaration:
```
function AddUnit(const AUnitName: string) : TTarget
function AddUnit(const AUnitName: string; const OSes: TOSes) : TTarget
function AddUnit(const AUnitName: string; const CPUs: TCPUS;
                 const OSes: TOSes) : TTarget
```

Visibility: public

Description: Creates and adds a new unit target with the specified name to the target collection. The unit will be compiled as a regular Pascal unit and can optionally be restricted to specific CPU architectures and operating systems.

See also: AddProgram (1062), AddLibrary (1063), AddImplicitUnit (1062)

29.54.6 TTargets.AddImplicitUnit

Synopsis: Adds an implicit unit target to the collection

Declaration:
```
function AddImplicitUnit(const AUnitName: string; InstallUnit: Boolean)
                         : TTarget
function AddImplicitUnit(const AUnitName: string; const OSes: TOSes;
                         InstallUnit: Boolean) : TTarget
function AddImplicitUnit(const AUnitName: string; const CPUs: TCPUS;
                         InstallUnit: Boolean) : TTarget
function AddImplicitUnit(const AUnitName: string; const CPUs: TCPUS;
                         const OSes: TOSes; InstallUnit: Boolean)
                         : TTarget
```

Visibility: public

Description: Creates and adds a new implicit unit target that is compiled automatically as a dependency without explicit reference in the build script.

Implicit units are typically used for helper modules and internal dependencies that do not need separate installation or distribution.

29.54.7 TTargets.AddProgram

Synopsis: Adds a program target to the collection

Declaration:
```
function AddProgram(const AProgramName: string) : TTarget
function AddProgram(const AProgramName: string; const OSes: TOSes)
                    : TTarget
```

```
function AddProgram(const AProgramName: string; const CPUs: TCPUS;
                    const OSes: TOSes) : TTarget
```

Visibility: public

Description: Creates and adds a new program target with the specified name to the target collection. The program will be compiled as an executable and can optionally be restricted to specific CPU architectures and operating systems.

See also: AddUnit (1062), AddLibrary (1063), ttProgram (930)

29.54.8 TTargets.AddLibrary

Synopsis: Adds a library target to the collection

Declaration:
```
function AddLibrary(const ALibraryName: string) : TTarget
function AddLibrary(const ALibraryName: string; const OSes: TOSes)
                     : TTarget
function AddLibrary(const ALibraryName: string; const CPUs: TCPUS;
                    const OSes: TOSes) : TTarget
```

Visibility: public

Description: Creates and adds a new library target with the specified name to the target collection. The library will be compiled as a shared library and can optionally be restricted to specific CPU architectures and operating systems.

See also: AddUnit (1062), AddProgram (1062), ttSharedLibrary (930)

29.54.9 TTargets.AddExampleUnit

Synopsis: Adds an example unit target to the collection

Declaration:
```
function AddExampleUnit(const AUnitName: string) : TTarget
function AddExampleUnit(const AUnitName: string; const OSes: TOSes)
                         : TTarget
function AddExampleUnit(const AUnitName: string; const CPUs: TCPUS;
                        const OSes: TOSes) : TTarget
```

Visibility: public

Description: Creates and adds a new example unit target that demonstrates usage patterns and serves as sample code for package functionality.

Example units are typically installed separately from main package units and provide documentation through working code examples.

29.54.10 TTargets.AddExampleProgram

Synopsis: Adds an example program target to the collection

Declaration:
```
function AddExampleProgram(const AProgramName: string) : TTarget
function AddExampleProgram(const AProgramName: string;
                           const OSes: TOSes) : TTarget
function AddExampleProgram(const AProgramName: string;
                           const CPUs: TCPUS; const OSes: TOSes)
                            : TTarget
```

Visibility: public

Description: Creates and adds a new example program target that demonstrates package functionality through executable sample applications.

Example programs provide working demonstrations of package features and serve as practical documentation for developers.

29.54.11 TTargets.GetEnumerator

Synopsis: Gets enumerator for iterating build targets

Declaration: `function GetEnumerator : TTargetEnumerator`

Visibility: public

Description: Returns a TTargetEnumerator (1060) object that enables for-in loop iteration through the build targets collection.

This method enables convenient enumeration of all build targets using modern Pascal iteration syntax.

29.54.12 TTargets.Targets

Synopsis: Access to build target by name

Declaration: `Property Targets[AName: string]: TTarget; default`

Visibility: public

Access: Read

Description: Provides named access to TTarget (1049) objects within the collection using the target's name as the key.

This property enables lookup and retrieval of specific build targets by their configured names.

29.54.13 TTargets.TargetItems

Synopsis: Indexed access to build targets in the collection

Declaration: `Property TargetItems[Index: Integer]: TTarget`

Visibility: public

Access: Read,Write

Description: Provides indexed read-only access to TTarget (1049) objects within the collection using zero-based array indexing.

This default property enables array-style access to build targets for iteration and processing.

29.55 TValueItem

29.55.1 Description

`TValueItem` represents a dictionary entry that stores a string value with an associated key. This class is used in template processing and variable substitution systems within the fpmake build system.

29.55.2 Method overview

Page	Method	Description
1065	Create	Creates value item with specified value

29.55.3 TValueItem.Create

Synopsis: Creates value item with specified value

Declaration: `constructor Create(AValue: string)`

Visibility: default

Description: `Create` initializes a new `TValueItem` instance with the specified string value. This constructor stores the provided value for later retrieval and use in template substitution.

The created value item can be added to dictionaries or collections for dynamic configuration management.

Chapter 30

Reference for unit 'fpobserver'

30.1 Used units

Table 30.1: Used units by unit 'fpobserver'

Name	Page
Classes	??
Contnrs	284
System	??
sysutils	??
TypInfo	??

30.2 Overview

The `fpobserver` unit provides a complete implementation of the Observer pattern for Free Pascal applications. It includes classes and interfaces for creating observable objects that can notify multiple observers when their state changes.

The unit contains the `TObservedHook` class for adding observer support to any object, the `TBaseMediator` hierarchy for automatic data binding between model and view objects, and mediator management classes for registering and finding appropriate mediators.

30.3 Constants, types and variables

30.3.1 Types

```
TMediatingEvent = procedure(Sender: TObject; var Handled: Boolean
  )
                                of object
```

The `TMediatingEvent` procedure type defines the signature for events that occur during data transfer operations in mediators. The event allows custom handling of data transfer between subject and view objects.

The `Handled` parameter can be set to `True` to indicate that the event handler has performed the data transfer, preventing the default transfer mechanism from executing.

```
TMediatorClass = Class of TBaseMediator
```

The `TMediatorClass` type is a class reference that points to TBaseMediator (1070) or its descendants. It is used in mediator registration and management to specify mediator classes without creating instances.

```
TOnBeforeSetupField = procedure(AObject: TObject;
  const AFieldName: string;
  var AValue: string) of object
```

The `TOnBeforeSetupField` procedure type defines the signature for events that allow formatting of field values before they are written to list view captions or items. This event provides access to the object, field name, and value that can be modified.

This event is meant to provide the possibility to apply custom formatting for dates, numbers, or other data types.

30.4 Procedures and functions

30.4.1 MediatorError

Synopsis: Raises formatted mediator error with context information

Declaration:
```
procedure MediatorError(Sender: TObject; const Msg: string);  Overload
procedure MediatorError(Sender: TObject; const Fmt: string;
                        Args: Array of const);  Overload
```

Visibility: default

Description: The `MediatorError` procedure raises an EMediator (1068) exception with detailed context information. It formats the error message with information about the sender object, including mediator details, view component names, and subject information where available.

Multiple overloads support both direct message strings and format string with arguments.

See also: EMediator (1068)

30.4.2 MediatorManager

Synopsis: Returns the global singleton mediator manager instance

Declaration:
```
function MediatorManager : TMediatorManager
```

Visibility: default

Description: The `MediatorManager` function returns the global singleton instance of the mediator manager. If the instance doesn't exist yet, it creates one automatically.

This function provides convenient access to the global mediator manager for registering mediators and finding appropriate mediator definitions.

This function is not thread-safe, so make sure to call it once in the main thread so the instance is created once.

See also: TMediatorManager (1085)

30.5 EMediator

30.5.1 Description

The `EMediator` class is the exception type raised by mediator-related operations when errors occur. It provides specific error reporting for mediator configuration and operation problems.

30.6 TBaseListMediator

30.6.1 Description

The `TBaseListMediator` class extends TBaseMediator (1070) to handle collections of objects. It creates individual item mediators for each object in the list and manages their lifecycle automatically.

The subject must be a `TObjectList`, `TList`, or `TCollection`. The mediator responds to list changes by creating, updating, or destroying item mediators as needed. T his class is abstract and must be subclassed to provide specific GUI integration.

Each item in the list is handled by a TListItemMediator (1077) instance, allowing individual objects to have their own observer relationships and data binding behavior.

See also: TBaseMediator (1070)

30.6.2 Method overview

Page	Method	Description
1069	CompositeMediator	Returns True indicating this is a composite mediator
1068	Create	Creates a base list mediator instance
1068	Destroy	Destroys the list mediator and all item mediators
1069	ObservedChanged	Handles changes in the observed list objects

30.6.3 Property overview

Page	Properties	Access	Description
1069	OnBeforeSetupField	rw	Event for formatting field values in all item mediators
1069	SelectedObject	rw	Currently selected object in the list

30.6.4 TBaseListMediator.Create

Synopsis: Creates a base list mediator instance

Declaration: `constructor Create(AOwner: TComponent); Override`

Visibility: public

Description: The `Create` constructor initializes the list mediator with an empty item mediator collection and sets the default `ViewPropertyName` to 'Caption'. The mediator is initially inactive.

30.6.5 TBaseListMediator.Destroy

Synopsis: Destroys the list mediator and all item mediators

Declaration: `destructor Destroy; Override`

Visibility: public

Description: The `Destroy` destructor cleans up all item mediators and frees the mediator collection.

30.6.6 TBaseListMediator.CompositeMediator

Synopsis: Returns True indicating this is a composite mediator

Declaration:
```
class function CompositeMediator : Boolean;  Override
```

Visibility: public

Description: The `CompositeMediator` method returns `True` for list mediators, indicating they handle multiple objects rather than single property mediation. This affects how the mediator manager registers and selects mediators.

Composite mediators work without specific property names and handle entire object relationships rather than individual property synchronization.

30.6.7 TBaseListMediator.ObservedChanged

Synopsis: Handles changes in the observed list objects

Declaration:
```
procedure ObservedChanged(ASender: TObject;
                          Operation: TFPObservedOperation)
```

Visibility: public

Description: The `ObservedChanged` method responds to changes in the list by creating mediators for added items, removing mediators for deleted items, or rebuilding the entire mediator collection when necessary.

30.6.8 TBaseListMediator.SelectedObject

Synopsis: Currently selected object in the list

Declaration:
```
Property SelectedObject : TObject
```

Visibility: public

Access: Read,Write

Description: The `SelectedObject` property provides access to the currently selected object in the list. The base implementation returns `nil`; descendants must override the getter and setter to provide meaningful selection support.

This property enables two-way binding between the GUI selection state and the object model.

30.6.9 TBaseListMediator.OnBeforeSetupField

Synopsis: Event for formatting field values in all item mediators

Declaration:
```
Property OnBeforeSetupField : TOnBeforeSetupField
```

Visibility: published

Access: Read,Write

Description: The `OnBeforeSetupField` event allows customization of how field values are formatted before display. It is automatically propagated to all item mediators in the collection.

This event is particularly useful for formatting dates, numbers, or other data types for consistent display across all list items.

30.7 TBaseMediator

30.7.1 Description

The `TBaseMediator` class provides automatic data binding between model objects (subjects) and view components. It implements the observer pattern to automatically synchronize data when either the model or view changes.

The mediator supports bidirectional data binding, read-only modes, and custom data transfer events. Subjects must implement the `IFPObserved` interface, while views must have published properties that can be accessed by name.

This class serves as the base class for more specialized mediators like list mediators and component mediators, providing common functionality for property validation, observer management, and data transfer coordination.

See also: TObservedHook (1087), TComponentMediator (1076)

30.7.2 Interfaces overview

Page	Interfaces	Description
??	IFPObserver	

30.7.3 Method overview

Page	Method	Description
1072	CompositeMediator	Indicates whether this is a composite mediator
1071	Destroy	Destroys the mediator and cleans up observer attachments
1071	ObjectToView	Copies data from subject object to view component
1071	ViewClass	Returns the minimum view class this mediator can handle
1071	ViewToObject	Copies data from view component to subject object

30.7.4 Property overview

Page	Properties	Access	Description
1073	Active	rw	Enables or disables automatic data synchronization
1074	OnObjectToView	rw	Event triggered when copying data from subject to view
1074	OnViewToObject	rw	Event triggered when copying data from view to subject
1073	ReadOnly	rw	Prevents data transfer from view to subject when True
1072	Subject	rw	Model object implementing IFPObserved interface
1073	SubjectPropertyName	rw	Name of property in subject object to mediate
1072	Valuelist	rw	Optional list of values for list-based mediators
1072	View	r	View component for data display and editing

30.7.5 TBaseMediator.Destroy

Synopsis: Destroys the mediator and cleans up observer attachments

Declaration: `destructor Destroy;  Override`

Visibility: public

Description: The `Destroy` destructor ensures proper cleanup by detaching the mediator from both the subject and value list objects.

30.7.6 TBaseMediator.ObjectToView

Synopsis: Copies data from subject object to view component

Declaration: `procedure ObjectToView`

Visibility: public

Description: The `ObjectToView` method transfers data from the subject's property to the view's property. It first checks for custom handling via the OnObjectToView (1074) event, then performs the default property copying if it was not handled in the event.

This method is called automatically when the subject changes, or can be called manually to force a data refresh.

See also: ViewToObject (1071), OnObjectToView (1074)

30.7.7 TBaseMediator.ViewToObject

Synopsis: Copies data from view component to subject object

Declaration: `procedure ViewToObject`

Visibility: public

Description: The `ViewToObject` method transfers data from the view's property to the subject's property. It first checks for custom handling via the OnViewToObject (1074) event, then performs the default property copying if not was not handled in the event.

This method is typically called when the view component changes, unless the mediator is in read-only mode.

See also: ObjectToView (1071), OnViewToObject (1074)

30.7.8 TBaseMediator.ViewClass

Synopsis: Returns the minimum view class this mediator can handle

Declaration: `class function ViewClass : TClass;  Virtual`

Visibility: public

Description: The `ViewClass` class method returns the base class that view objects must inherit from to be compatible with this mediator. Descendant mediators override this to specify more specific view class requirements.

The default implementation returns `TObject`, indicating that any object can serve as a view.

30.7.9 TBaseMediator.CompositeMediator

Synopsis: Indicates whether this is a composite mediator

Declaration: `class function CompositeMediator : Boolean; Virtual`

Visibility: public

Description: The `CompositeMediator` class method returns `False` for base mediators, indicating they handle single property mediation. List mediators and other composite mediators override this to return `True`.

This information is used by the mediator manager when automatically selecting appropriate mediators for different scenarios.

30.7.10 TBaseMediator.Subject

Synopsis: Model object implementing IFPObserved interface

Declaration: `Property Subject : TObject`

Visibility: public

Access: Read,Write

Description: The `Subject` property holds the model object whose data will be mediated to the view. The object must implement the `IFPObserved` interface to support change notifications.

When set, the mediator automatically attaches as an observer and performs initial data synchronization if active.

See also: View (1072), ViewClass (1071), ViewPropertyName (1070)

30.7.11 TBaseMediator.View

Synopsis: View component for data display and editing

Declaration: `Property View : TObject`

Visibility: public

Access: Read

Description: The `View` property provides access to the view component that displays and allows editing of the subject's data. The base class returns `nil`; descendant classes override `GetView` to return their specific view component.

The view must have a published property matching ViewPropertyName (1070) for data binding to work correctly.

See also: Subject (1072), ViewClass (1071), ViewPropertyName (1070)

30.7.12 TBaseMediator.Valuelist

Synopsis: Optional list of values for list-based mediators

Declaration: `Property Valuelist : TObjectList`

Visibility: public

Access: Read,Write

Description: The `Valuelist` property holds an optional list of values used by mediators that work with dynamic value lists, such as combobox or listbox mediators. The list must implement `IFPObserved` for change notifications.

When set, the mediator observes the value list for changes and updates the view accordingly.

See also: Subject (1072), SubjectPropertyName (1072), ViewPropertyName (1070)

30.7.13 TBaseMediator.SubjectPropertyName

Synopsis: Name of property in subject object to mediate

Declaration: `Property SubjectPropertyName : string`

Visibility: published

Access: Read,Write

Description: The `SubjectPropertyName` property specifies which published property of the subject object will be synchronized with the view. The property must exist and be published for the mediation to work correctly.

Setting this property triggers validation and may cause immediate data synchronization if the mediator is active.

See also: Subject (1072), ViewPropertyName (1070)

30.7.14 TBaseMediator.Active

Synopsis: Enables or disables automatic data synchronization

Declaration: `Property Active : Boolean`

Visibility: published

Access: Read,Write

Description: The `Active` property controls whether the mediator performs automatic data synchronization. When `True`, the mediator responds to subject changes and transfers data between subject and view.

Setting this to `False` stops all automatic synchronization, though manual calls to `ObjectToView` and `ViewToObject` still work.

See also: ReadOnly (1073)

30.7.15 TBaseMediator.ReadOnly

Synopsis: Prevents data transfer from view to subject when True

Declaration: `Property ReadOnly : Boolean`

Visibility: published

Access: Read,Write

Description: The `ReadOnly` property controls the direction of data transfer. When `True`, data flows only from subject to view, preventing user input from modifying the subject object.

This is useful for display-only scenarios where the view should reflect the subject's state but not allow modifications.

See also: Active (1073)

30.7.16 TBaseMediator.OnViewToObject

Synopsis: Event triggered when copying data from view to subject

Declaration: `Property OnViewToObject : TMediatingEvent`

Visibility: published

Access: Read,Write

Description: The `OnViewToObject` event is triggered before the default view-to-subject data transfer. Setting the `Handled` parameter to `True` prevents the default transfer, allowing custom data handling.

This event is useful for implementing custom validation, data transformation, or special handling during user input processing.

See also: OnObjectToView (1074), Subject (1072), ViewPropertyName (1070)

30.7.17 TBaseMediator.OnObjectToView

Synopsis: Event triggered when copying data from subject to view

Declaration: `Property OnObjectToView : TMediatingEvent`

Visibility: published

Access: Read,Write

Description: The `OnObjectToView` event is triggered before the default subject-to-view data transfer. Setting the `Handled` parameter to `True` prevents the default transfer, allowing custom data presentation.

This event is useful for implementing custom formatting, data transformation, or special display logic when updating the view.

See also: OnViewToObject (1074), Subject (1072), ViewPropertyName (1070)

30.8 TColumnsListItemMediator

30.8.1 Description

The `TColumnsListItemMediator` class extends TListItemMediator (1077) to provide access to the parent mediator's field information collection. This allows item mediators to access column configuration for multi-column display scenarios.

See also: TListItemMediator (1077)

30.8.2 Property overview

Page	Properties	Access	Description
1074	FieldsInfo	r	Reference to parent mediator's field configuration

30.8.3 TColumnsListItemMediator.FieldsInfo

Synopsis: Reference to parent mediator's field configuration

Declaration: `Property FieldsInfo : TMediatorFieldInfoList`

Visibility: published

Access: Read

Description: The `FieldsInfo` property provides access to the parent `TColumnsListMediator`'s field information collection. This allows the item mediator to access column definitions for data formatting and display.

30.9 TColumnsListMediator

30.9.1 Description

The `TColumnsListMediator` class extends TBaseListMediator (1068) to support multi-column list displays. It manages a collection of field information objects that define the properties, captions, widths, and alignments for each column.

This class provides the foundation for list controls that display multiple properties of each object in a tabular format, such as list views or grids.

See also: TBaseListMediator (1068)

30.9.2 Method overview

Page	Method	Description
1075	Create	Creates a columns list mediator with field info collection
1075	Destroy	Destroys the columns mediator and field info collection

30.9.3 Property overview

Page	Properties	Access	Description
1076	DisplayNames	rw	Convenient string format for setting column properties
1076	FieldsInfo	rw	Collection of field information for column display configuration

30.9.4 TColumnsListMediator.Create

Synopsis: Creates a columns list mediator with field info collection

Declaration: `constructor Create(AOwner: TComponent); Override`

Visibility: public

Description: The `Create` constructor initializes the columns mediator with an empty field information collection and sets the default `SubjectPropertyName` to 'Caption'.

30.9.5 TColumnsListMediator.Destroy

Synopsis: Destroys the columns mediator and field info collection

Declaration: `destructor Destroy; Override`

Visibility: public

Description: The `Destroy` destructor cleans up the field information collection along with all item mediators, ensuring complete cleanup of the mediator structure.

30.9.6 TColumnsListMediator.DisplayNames

Synopsis: Convenient string format for setting column properties

Declaration: `Property DisplayNames : string`

Visibility: public

Access: Read,Write

Description: The `DisplayNames` property provides a convenient way to configure all columns using a semicolon-separated string format. Setting this property parses the string and updates the field information collection.

This property is particularly useful for quick configuration and design-time setup of column layouts.

See also: FieldsInfo (1076)

30.9.7 TColumnsListMediator.FieldsInfo

Synopsis: Collection of field information for column display configuration

Declaration: `Property FieldsInfo : TMediatorFieldInfoList`

Visibility: published

Access: Read,Write

Description: The `FieldsInfo` property contains the collection of field information objects that define how each column should be displayed. This collection can be modified at design time or runtime to configure the column layout.

Changes to this collection are monitored and may trigger column recreation if the mediator is active.

See also: DisplayNames (1076)

30.10 TComponentMediator

30.10.1 Description

The `TComponentMediator` class provides mediation capabilities for any `TComponent`-based view. It extends TBaseMediator (1070) to work with components that can be set at design time or runtime.

This mediator is useful for creating flexible data binding scenarios where the specific view component type is not known at compile time, or when working with various types of input controls. You can also create descendants from this class which introduce more specific behaviour

See also: TBaseMediator (1070)

30.10.2 Property overview

Page	Properties	Access	Description
1077	ViewComponent	rw	Component serving as the view in the mediation
1077	ViewPropertyName		Name of property in view component to mediate

30.10.3 TComponentMediator.ViewComponent

Synopsis: Component serving as the view in the mediation

Declaration: `Property ViewComponent : TComponent`

Visibility: published

Access: Read,Write

Description: The `ViewComponent` property specifies which component will serve as the view for data binding. Setting this property establishes free notification handling and triggers view change processing.

See also: ViewPropertyName (1077)

30.10.4 TComponentMediator.ViewPropertyName

Synopsis: Name of property in view component to mediate

Declaration: `Property ViewPropertyName :`

Visibility: published

Access:

Description: The `ViewPropertyName` property specifies which published property of the view component will be synchronized with the subject. This property is published to allow configuration in the Object Inspector.

The specified property must exist and be published in the view component for the mediation to function correctly.

See also: ViewComponent (1077)

30.11 TListItemMediator

30.11.1 Description

The `TListItemMediator` class handles individual objects within list mediators. Each item in a list gets its own item mediator that manages the observer relationship and data synchronization for that specific object.

Item mediators are created and managed automatically by their parent list mediator and coordinate with it for activation state and field formatting events.

See also: TBaseListMediator (1068)

30.11.2 Interfaces overview

Page	Interfaces	Description
??	IFPObserver	

30.11.3 Method overview

Page	Method	Description
1078	Destroy	Destroys the list item mediator and detaches from subject
1078	MaybeObjectToView	Conditionally updates view with data from subject

30.11.4 Property overview

Page	Properties	Access	Description
1079	Active	r	Indicates if the parent list mediator is active
1078	OnBeforeSetupField	rw	Event for formatting field values before display
1078	Subject	rw	Individual object being mediated by this item mediator

30.11.5 TListItemMediator.Destroy

Synopsis: Destroys the list item mediator and detaches from subject

Declaration: `destructor Destroy;  Override`

Visibility: public

Description: The `Destroy` destructor ensures proper cleanup by detaching from the subject object, p reventing observer notifications to a destroyed mediator.

30.11.6 TListItemMediator.MaybeObjectToView

Synopsis: Conditionally updates view with data from subject

Declaration: `procedure MaybeObjectToView`

Visibility: public

Description: The `MaybeObjectToView` method checks if both the subject is assigned and the parent mediator is active before triggering a view update.

30.11.7 TListItemMediator.OnBeforeSetupField

Synopsis: Event for formatting field values before display

Declaration: `Property OnBeforeSetupField : TOnBeforeSetupField`

Visibility: public

Access: Read,Write

Description: The `OnBeforeSetupField` event allows custom formatting of field values before they are displayed. This event is typically set by the parent list mediator to ensure consistent formatting across all items.

30.11.8 TListItemMediator.Subject

Synopsis: Individual object being mediated by this item mediator

Declaration: `Property Subject : TObject`

Visibility: public

Access: Read,Write

Description: The `Subject` property holds the specific object that this item mediator is responsible for. The object must implement `IFPObserved` for proper notification handling.

When set, the mediator automatically establishes observer relationships and may trigger view updates if active.

30.11.9 TListItemMediator.Active

Synopsis: Indicates if the parent list mediator is active

Declaration:
```
Property Active : Boolean
```

Visibility: public

Access: Read

Description: The `Active` property returns the active state of the parent list mediator. Item mediators don't have their own activation state but inherit it from their parent.

30.12 TMediatorDef

30.12.1 Description

The `TMediatorDef` class represents a mediator definition used by the mediator manager to automatically select appropriate mediator classes for given subject/GUI combinations.

Each definition specifies the mediator class, minimum subject class, and either specific property names or property types that the mediator can handle. The manager uses this information to find the best matching mediator for automatic mediation scenarios.

See also: Handles (1079)

30.12.2 Method overview

Page	Method	Description
1080	BetterMatch	Compares this definition against another for best match selection
1079	Handles	Determines if this definition can handle the given subject/GUI combination

30.12.3 Property overview

Page	Properties	Access	Description
1080	MediatorClass	rw	Class reference for the mediator implementation
1080	MinSubjectClass	rw	Minimum subject class that this mediator can handle
1081	PropertyName	rw	Specific property name this mediator handles
1080	PropertyTypes	rw	Set of property types this mediator can handle

30.12.4 TMediatorDef.Handles

Synopsis: Determines if this definition can handle the given subject/GUI combination

Declaration:
```
function Handles(ASubject: TObject; AGui: TComponent;
                 APropInfo: PPropInfo) : Boolean
```

Visibility: public

Description: The `Handles` method evaluates whether this mediator definition can handle the specified combination of subject object, GUI component, and property information. It checks class compatibility and property requirements.

This method is used by the mediator manager during the selection process to filter potentially suitable mediators.

See also: TMediatorDef.BetterMatch (1080)

30.12.5 TMediatorDef.BetterMatch

Synopsis: Compares this definition against another for best match selection

Declaration: `function BetterMatch(M: TMediatorDef) : Boolean`

Visibility: public

Description: The `BetterMatch` method compares this definition against another to determine which provides a better match for a given scenario. It considers property name specificity, class hierarchies, and mediator types.

This comparison logic ensures that the most specific and appropriate mediator is selected when multiple mediators could handle the same situation.

See also: FindDefFor (1086)

30.12.6 TMediatorDef.MediatorClass

Synopsis: Class reference for the mediator implementation

Declaration: `Property MediatorClass : TMediatorClass`

Visibility: public

Access: Read,Write

Description: The `MediatorClass` property holds the class reference for the mediator that will be instantiated when this definition is selected. This must be a descendant of `TBaseMediator`.

See also: TBaseMediator (1070)

30.12.7 TMediatorDef.MinSubjectClass

Synopsis: Minimum subject class that this mediator can handle

Declaration: `Property MinSubjectClass : TClass`

Visibility: public

Access: Read,Write

Description: The `MinSubjectClass` property specifies the minimum class that subject objects must inherit from for this mediator to be applicable. This allows mediators to be restricted to specific object hierarchies.

See also: TBaseMediator.Subject (1072)

30.12.8 TMediatorDef.PropertyTypes

Synopsis: Set of property types this mediator can handle

Declaration: `Property PropertyTypes : TTypeKinds`

Visibility: public

Access: Read,Write

Description: The `PropertyTypes` property defines the set of property types that this mediator can handle. This is used when the mediator should work with properties of specific types rather than specific property names.

30.12.9 TMediatorDef.PropertyName

Synopsis: Specific property name this mediator handles

Declaration: `Property PropertyName : string`

Visibility: public

Access: Read,Write

Description: The `PropertyName` property specifies a specific property name that this mediator is designed to handle. When set, the mediator will only be selected for properties with this exact name.

30.13 TMediatorDefs

30.13.1 Description

The `TMediatorDefs` class is a collection that holds mediator definitions for the mediator manager. It provides methods for adding new definitions and accessing existing ones by index.

The collection is used internally by the mediator manager to store and organize all registered mediator definitions, so normally you should not use this class directly.

See also: TMediatorManager (1085)

30.13.2 Method overview

Page	Method	Description
1081	AddDef	Adds a new mediator definition to the collection

30.13.3 Property overview

Page	Properties	Access	Description
1081	Defs	rw	Indexed access to individual mediator definitions

30.13.4 TMediatorDefs.AddDef

Synopsis: Adds a new mediator definition to the collection

Declaration: `function AddDef : TMediatorDef`

Visibility: public

Description: The `AddDef` method creates a new mediator definition and adds it to the collection. The returned definition can then be configured with the appropriate mediator class, subject class, and property requirements.

See also: TMediatorManager.RegisterMediator (1087)

30.13.5 TMediatorDefs.Defs

Synopsis: Indexed access to individual mediator definitions

Declaration: `Property Defs[Index: Integer]: TMediatorDef; default`

Visibility: public

Access: Read,Write

Description: The `Defs` property provides indexed access to mediator definitions in the collection. This is the default property, allowing array-style access syntax.

See also: FindDefFor (1086)

30.14 TMediatorFieldInfo

30.14.1 Description

The `TMediatorFieldInfo` class defines the properties of a column in list mediators that support multiple columns. It specifies the property name, display caption, width, and alignment for each column.

Field info objects are typically managed within a TMediatorFieldInfoList (1084) collection and can be configured through string parsing or individual property settings.

See also: TMediatorFieldInfoList (1084)

30.14.2 Method overview

Page	Method	Description
1082	Assign	Copies properties from another field info object

30.14.3 Property overview

Page	Properties	Access	Description
1084	Alignment	rw	Text alignment for the column content
1082	AsString	rw	String representation of field info configuration
1083	Caption	rw	Display caption for the column header
1083	PropertyName	rw	Name of property to display in this column
1083	Width	rw	Width of the column in display units

30.14.4 TMediatorFieldInfo.Assign

Synopsis: Copies properties from another field info object

Declaration: `procedure Assign(Source: TPersistent);  Override`

Visibility: public

Description: The `Assign` method copies all properties from another `TMediatorFieldInfo` object, including property name, caption, width, and alignment settings.

30.14.5 TMediatorFieldInfo.AsString

Synopsis: String representation of field info configuration

Declaration: `Property AsString : string`

Visibility: public

Access: Read,Write

Description: The `AsString` property provides a convenient way to get or set all field properties using a single string. The string must be in the format `'PropertyName|Alignment|Width|Caption'`. This format is used for batch configuration and serialization.

Setting this property parses the string and updates all individual properties accordingly.

See also: Caption (1083), PropertyName (1083), Width (1083), Alignment (1084)

30.14.6 TMediatorFieldInfo.Caption

Synopsis: Display caption for the column header

Declaration: `Property Caption : string`

Visibility: published

Access: Read,Write

Description: The `Caption` property specifies the text displayed in the column header. If not explicitly set, it defaults to the property name, providing a reasonable fallback for display purposes.

See also: PropertyName (1083), Width (1083), Alignment (1084)

30.14.7 TMediatorFieldInfo.PropertyName

Synopsis: Name of property to display in this column

Declaration: `Property PropertyName : string`

Visibility: published

Access: Read,Write

Description: The `PropertyName` property specifies which property of the list items will be displayed in this column. The property must be published and accessible for the column to display data correctly.

See also: Caption (1083), Width (1083), Alignment (1084)

30.14.8 TMediatorFieldInfo.Width

Synopsis: Width of the column in display units

Declaration: `Property Width : Integer`

Visibility: published

Access: Read,Write

Description: The `Width` property specifies the column width in pixels or other display units, depending on the specific GUI implementation. The default width is 50 units.

See also: Caption (1083), PropertyName (1083), Alignment (1084)

30.14.9 TMediatorFieldInfo.Alignment

Synopsis: Text alignment for the column content

Declaration:
```
Property Alignment : TAlignment
```

Visibility: published

Access: Read,Write

Description: The `Alignment` property specifies how text should be aligned within the column. It uses the standard `TAlignment` type supporting left, right, and center alignment.

See also: Caption (1083), PropertyName (1083), Width (1083)

30.15 TMediatorFieldInfoList

30.15.1 Description

The `TMediatorFieldInfoList` class manages a collection of TMediatorFieldInfo (1082) objects, providing methods for adding field configurations and accessing them by index.

The collection supports string-based batch configuration and provides notification to parent mediators when field configurations change.

See also: TMediatorFieldInfo (1082)

30.15.2 Method overview

Page	Method	Description
1084	AddFieldInfo	Adds a new field information item to the collection

30.15.3 Property overview

Page	Properties	Access	Description
1085	AsString	r	String representation of all field configurations
1085	FieldInfo	rw	Indexed access to field information items

30.15.4 TMediatorFieldInfoList.AddFieldInfo

Synopsis: Adds a new field information item to the collection

Declaration:
```
function AddFieldInfo : TMediatorFieldInfo;  Overload
function AddFieldInfo(const APropName: string; AFieldWidth: Integer)
                       : TMediatorFieldInfo;  Overload
function AddFieldInfo(const APropName: string; const ACaption: string;
                      AFieldWidth: Integer) : TMediatorFieldInfo
                      ;  Overload
function AddFieldInfo(const APropName: string; const ACaption: string;
                      AFieldWidth: Integer; AAlignment: TAlignment)
                       : TMediatorFieldInfo;  Overload
```

Visibility: public

Description: The `AddFieldInfo` method creates a new field information item and adds it to the collection. Multiple overloads are available to specify different combinations of property name, width, caption, and alignment.

The method returns the newly created field info object for further customization if needed.

30.15.5 TMediatorFieldInfoList.FieldInfo

Synopsis: Indexed access to field information items

Declaration: `Property FieldInfo[Index: Integer]: TMediatorFieldInfo; default`

Visibility: public

Access: Read,Write

Description: The `FieldInfo` property provides indexed access to individual field information objects in the collection. This is the default property, allowing array-style access syntax.

See also: AsString (1085)

30.15.6 TMediatorFieldInfoList.AsString

Synopsis: String representation of all field configurations

Declaration: `Property AsString : string`

Visibility: public

Access: Read

Description: The `AsString` property provides a semicolon-separated string containing all field configurations in the collection. This format is convenient for serialization and bulk configuration.

Each field is represented in the format 'PropertyName|Alignment|Width|Caption', with fields separated by semicolons.

See also: FieldInfo (1085)

30.16 TMediatorManager

30.16.1 Description

The `TMediatorManager` class provides centralized registration and lookup of mediator definitions. It allows registration of mediator classes with specific requirements and automatically finds the best matching mediator for given subject/GUI combinations.

The manager uses a collection of mediator definitions to evaluate compatibility and select the most appropriate mediator based on class hierarchies, property types, and property names.

30.16.2 Method overview

Page	Method	Description
1086	Create	Creates the mediator manager with empty definition collection
1086	Destroy	Destroys the manager and its definition collection
1086	FindDefFor	Finds the best mediator definition for given parameters
1087	RegisterMediator	Registers a new mediator class with the manager

30.16.3 Property overview

Page	Properties	Access	Description
1087	Defs	r	Collection of registered mediator definitions

30.16.4 TMediatorManager.Create

Synopsis: Creates the mediator manager with empty definition collection

Declaration:
```
constructor Create
```

Visibility: public

Description: The `Create` constructor initializes the mediator manager with an empty collection of mediator definitions. Mediator classes must be registered before the manager can provide automatic mediator selection.

See also: RegisterMediator (1087)

30.16.5 TMediatorManager.Destroy

Synopsis: Destroys the manager and its definition collection

Declaration:
```
destructor Destroy;  Override
```

Visibility: public

Description: The `Destroy` destructor cleans up the mediator definition collection, freeing all registered mediator definitions.

30.16.6 TMediatorManager.FindDefFor

Synopsis: Finds the best mediator definition for given parameters

Declaration:
```
function FindDefFor(ASubject: TObject; AGui: TComponent) : TMediatorDef
                    ;  Overload
function FindDefFor(ASubject: TObject; AGui: TComponent;
                    const APropName: string) : TMediatorDef;  Overload
function FindDefFor(ASubject: TObject; AGui: TComponent;
                    APropInfo: PPropInfo) : TMediatorDef;  Overload
```

Visibility: public

Description: The `FindDefFor` method searches through all registered mediator definitions to find the best match for the specified combination of subject object, GUI component, and optional property information.

The method evaluates each definition's compatibility and uses the `BetterMatch` comparison to select the most appropriate mediator. Multiple overloads support different search criteria including property names and property information.

See also: RegisterMediator (1087)

30.16.7 TMediatorManager.RegisterMediator

Synopsis: Registers a new mediator class with the manager

Declaration:
```
function RegisterMediator(MediatorClass: TMediatorClass;
                          MinSubjectClass: TClass) : TMediatorDef
                          ;  Overload
function RegisterMediator(MediatorClass: TMediatorClass;
                          MinSubjectClass: TClass;
                          const PropertyName: string) : TMediatorDef
                          ;  Overload
function RegisterMediator(MediatorClass: TMediatorClass;
                          MinSubjectClass: TClass;
                          PropertyTypes: TTypeKinds) : TMediatorDef
                          ;  Overload
```

Visibility: public

Description: The `RegisterMediator` method creates a new mediator definition and adds it to the manager's collection. Multiple overloads allow registration with different criteria including minimum subject class, property names, and property types.

The method returns the created definition, which can be further customized if needed.

30.16.8 TMediatorManager.Defs

Synopsis: Collection of registered mediator definitions

Declaration: `Property Defs : TMediatorDefs`

Visibility: public

Access: Read

Description: The `Defs` property provides access to the collection of all registered mediator definitions. This can be used for advanced scenarios that require direct access to the definition collection.

30.17 TObservedHook

30.17.1 Description

The `TObservedHook` class provides a simple way to add observer pattern support to any object. It implements the `IFPObserved` interface and manages a list of observers that implement the `IFPObserver` interface.

Objects can use this class as a helper to notify observers about changes, item additions, deletions, or custom events. The class handles attaching and detaching observers automatically and provides methods for different types of notifications.

This class is intended to be used as a component within other classes that need to support observation, rather than being used directly as a standalone object.

See also: IFPObserver (1066)

30.17.2 Interfaces overview

Page	Interfaces	Description
??	IFPObserved	

30.17.3 Method overview

Page	Method	Description
1089	AddItem	Notifies observers that an item was added
1089	Changed	Notifies all observers that the object has changed
1088	CreateSender	Creates an observer hook with a specified sender object
1090	CustomNotify	Sends custom notification to all observers
1089	DeleteItem	Notifies observers that an item was deleted
1088	Destroy	Destroys the observer hook and notifies all observers
1088	FPOAttachObserver	Attaches an observer implementing IFPObserver interface
1089	FPODetachObserver	Detaches an observer from the notification list
1090	FPONotifyObservers	Core method to notify all observers about operations

30.17.4 Property overview

Page	Properties	Access	Description
1090	Sender	r	Default sender object for notifications

30.17.5 TObservedHook.CreateSender

Synopsis: Creates an observer hook with a specified sender object

Declaration: `constructor CreateSender(ASender: TObject)`

Visibility: public

Description: The `CreateSender` constructor initializes the observer hook with a specific sender object that will be used in all notifications. If `ASender` is `nil`, the hook itself becomes the default sender.

The sender object is important because observers receive it as a parameter in their notification callbacks, allowing them to identify the source of the notification.

30.17.6 TObservedHook.Destroy

Synopsis: Destroys the observer hook and notifies all observers

Declaration: `destructor Destroy;  Override`

Visibility: public

Description: The `Destroy` destructor ensures proper cleanup by notifying all attached observers that the observed object is being destroyed.

This prevents observers from trying to access a destroyed object and gives them a chance to perform their own cleanup operations if needed.

30.17.7 TObservedHook.FPOAttachObserver

Synopsis: Attaches an observer implementing IFPObserver interface

Declaration: `procedure FPOAttachObserver(AObserver: TObject)`

Visibility: public

Description: The `FPOAttachObserver` method adds an observer to the notification list. The observer object must implement the `IFPObserver` interface or an exception will be raised.

Once attached, the observer will receive notifications about all operations performed on the observed object, including changes, item additions, deletions, and custom notifications.

Errors: The same observer can only be attached once; attempting to attach it multiple times will result in an EObserver (1066) Eexception.

See also: EObserver (1066), TObservedHook.FPODetachObserver (1089)

30.17.8 TObservedHook.FPODetachObserver

Synopsis: Detaches an observer from the notification list

Declaration: `procedure FPODetachObserver(AObserver: TObject)`

Visibility: public

Description: The `FPODetachObserver` method removes an observer from the notification list. The observer will no longer receive notifications about operations on the observed object.

If the observer is not currently attached, the method does nothing

See also: TObservedHook.FPOAttachObserver (1088)

30.17.9 TObservedHook.Changed

Synopsis: Notifies all observers that the object has changed

Declaration: `procedure Changed`

Visibility: public

Description: The `Changed` method sends a change notification to all attached observers using the `ooChange` operation.

Observers receive this notification through their `FPOObservedChanged` method and can respond by updating their display or performing other change-related operations.

30.17.10 TObservedHook.AddItem

Synopsis: Notifies observers that an item was added

Declaration: `procedure AddItem(AItem: TObject)`

Visibility: public

Description: The `AddItem` method sends an item addition notification to all observers using the `ooAddItem` operation. The added item is passed as the data parameter.

This notification is typically used by collection-like objects to inform observers when new items are added to the collection.

See also: DeleteItem (1089), Changed (1089), CustomNotify (1090), FPONotifyObservers (1090)

30.17.11 TObservedHook.DeleteItem

Synopsis: Notifies observers that an item was deleted

Declaration: `procedure DeleteItem(AItem: TObject)`

Visibility: public

Description: The `DeleteItem` method sends an item deletion notification to all observers using the `ooDeleteItem` operation. The deleted item is passed as the data parameter.

This notification is typically used by collection-like objects to inform observers when items are removed from the collection.

See also: AddItem (1089), Changed (1089), CustomNotify (1090), FPONotifyObservers (1090)

30.17.12 TObservedHook.CustomNotify

Synopsis: Sends custom notification to all observers

Declaration:
```
procedure CustomNotify(Data: Pointer)
```

Visibility: public

Description: The `CustomNotify` method sends a custom notification to all observers using the `ooCustom` operation. Optional data can be passed to provide context for the custom notification.

This method is useful for application-specific notifications that don't fit into the standard change, add item, or delete item categories.

See also: AddItem (1089), DeleteItem (1089), Changed (1089), FPONotifyObservers (1090)

30.17.13 TObservedHook.FPONotifyObservers

Synopsis: Core method to notify all observers about operations

Declaration:
```
procedure FPONotifyObservers(ASender: TObject;
                             AOperation: TFPObservedOperation;
                             Data: Pointer)
```

Visibility: public

Description: The `FPONotifyObservers` method is the central notification mechanism that calls the `FPOObservedChanged` method on all attached observers.

This method is used internally by all other notification methods and can be called directly for custom notification scenarios. The operation parameter specifies the type of change that occurred.

The method iterates through the observer list in reverse order to handle observers that might detach themselves during notification.

See also: AddItem (1089), DeleteItem (1089), Changed (1089), CustomNotify (1090)

30.17.14 TObservedHook.Sender

Synopsis: Default sender object for notifications

Declaration:
```
Property Sender : TObject
```

Visibility: public

Access: Read

Description: The `Sender` property contains the object that will be passed as the sender parameter to all observer notifications. This is typically the object that contains or uses the observer hook.

See also: AddItem (1089), DeleteItem (1089), Changed (1089), CustomNotify (1090), FPONotifyObservers (1090)

Chapter 31

Reference for unit 'fpTemplate'

31.1 Used units

Table 31.1: Used units by unit 'fpTemplate'

Name	Page
Classes	??
System	??
sysutils	??

31.2 Overview

fpTemplate implements a class TTemplateParser (1106) which can replace variables in a template text. The template variables are enclosed in delimiters. When a variable is encountered, it can be replaced by a string value. An event is called to get the value for a template variable. It does not contain loops or conditionals, but parametrized templates are possible.

The unit also contains TFPCustomTemplate (1099) and its descendant TFPTemplate (1103). Which is a basic implementation that uses the template parser to substitute variables in a text.

31.3 Using templates with parameters

`TTemplateParser` supports parametrized templates where tags can have parameters. To use parametrized templates, set `AllowTagParams` to `True` and use the `OnReplaceTag` event instead of `OnGetParam`. Parameters are enclosed in square brackets with the format `{TagName[-param1=value1-][-pa`

The following example demonstrates parametrized template usage:

```
program ParametrizedTemplateExample;

{$mode objfpc}{$h+}

uses
  fpTemplate, SysUtils, Classes;
```

```
type
  TEventHandler = class
    procedure ReplaceTagHandler(Sender: TObject; const TagString: string;
      TagParams: TStringList; out ReplaceText: string);
  end;

procedure TEventHandler.ReplaceTagHandler(Sender: TObject; const TagString: string;
  TagParams: TStringList; out ReplaceText: string);
var
  i: Integer;
  ParamName, ParamValue: string;
begin
  if TagString = 'FORMAT' then
  begin
    // Handle FORMAT tag with parameters
    ReplaceText := 'Formatted text';
    for i := 0 to TagParams.Count - 1 do
    begin
      ParamName := TagParams.Names[i];
      ParamValue := TagParams.ValueFromIndex[i];
      if ParamName = 'style' then
      begin
        if ParamValue = 'bold' then
          ReplaceText := '**' + ReplaceText + '**'
        else if ParamValue = 'italic' then
          ReplaceText := '*' + ReplaceText + '*';
      end
      else if ParamName = 'color' then
        ReplaceText := '[' + ParamValue + ']' + ReplaceText + '[/' + ParamValue + ']
    end;
  end
  else if TagString = 'REPEAT' then
  begin
    // Handle REPEAT tag with count parameter
    ReplaceText := '';
    for i := 0 to TagParams.Count - 1 do
    begin
      ParamName := TagParams.Names[i];
      ParamValue := TagParams.ValueFromIndex[i];
      if ParamName = 'count' then
      begin
        ReplaceText := StringOfChar('*', StrToIntDef(ParamValue, 1));
        Break;
      end;
    end;
  end
  else if TagString = 'TABLE' then
  begin
    // Handle TABLE tag with rows and cols parameters
    ReplaceText := 'Table';
    for i := 0 to TagParams.Count - 1 do
    begin
      ParamName := TagParams.Names[i];
      ParamValue := TagParams.ValueFromIndex[i];
```

```
      if ParamName = 'rows' then
        ReplaceText := ReplaceText + '(' + ParamValue + ' rows'
      else if ParamName = 'cols' then
        ReplaceText := ReplaceText + ', ' + ParamValue + ' cols)';
    end;
  end
  else
    ReplaceText := 'Unknown tag: ' + TagString;
end;

var
  Parser: TTemplateParser;
  Handler: TEventHandler;
  Template, Result: string;
begin
  Parser := TTemplateParser.Create;
  Handler := TEventHandler.Create;
  try
    // Enable parametrized templates
    Parser.AllowTagParams := True;

    // Assign tag replacement handler
    Parser.OnReplaceTag := @Handler.ReplaceTagHandler;

    // Define template text with parametrized tags
    Template := 'This is {FORMAT[-style=bold-]} text with {FORMAT[-style=italic-][-c
                'Here are some {REPEAT[-count=5-]} stars.' + LineEnding +
                'And here is a {TABLE[-rows=3-][-cols=4-]} definition.';

    // Parse the template
    Result := Parser.ParseString(Template);

    WriteLn(Result);
  finally
    Handler.Free;
    Parser.Free;
  end;
end.
```

This will output:

```
This is **Formatted text** text with [red]*Formatted text*[/red] formatting.
Here are some ***** stars.
And here is a Table(3 rows, 4 cols) definition.
```

You can also use parametrized templates with `TFPTemplate`:

```
program ParametrizedTFPTemplateExample;

{$mode objfpc}{$h+}

uses
  fpTemplate, SysUtils, Classes;
```

```
type
  TEventHandler = class
    procedure ReplaceTagHandler(Sender: TObject; const TagString: string;
      TagParams: TStringList; out ReplaceText: string);
  end;

procedure TEventHandler.ReplaceTagHandler(Sender: TObject; const TagString: string;
  TagParams: TStringList; out ReplaceText: string);
var
  i: Integer;
  ParamName, ParamValue: string;
begin
  if TagString = 'LINK' then
  begin
    // Handle LINK tag with url and text parameters
    ReplaceText := '#';
    for i := 0 to TagParams.Count - 1 do
    begin
      ParamName := TagParams.Names[i];
      ParamValue := TagParams.ValueFromIndex[i];
      if ParamName = 'url' then
        ReplaceText := '<a href="' + ParamValue + '"|>'
      else if ParamName = 'text' then
        ReplaceText := ReplaceText + ParamValue + '</a>';
    end;
  end
  else if TagString = 'IMG' then
  begin
    // Handle IMG tag with src and alt parameters
    ReplaceText := '<img';
    for i := 0 to TagParams.Count - 1 do
    begin
      ParamName := TagParams.Names[i];
      ParamValue := TagParams.ValueFromIndex[i];
      ReplaceText := ReplaceText + ' ' + ParamName + '="' + ParamValue + '"';
    end;
    ReplaceText := ReplaceText + ' />';
  end
  else
    ReplaceText := '';
end;

var
  Template: TFPTemplate;
  Handler: TEventHandler;
  Result: string;
begin
  Template := TFPTemplate.Create;
  Handler := TEventHandler.Create;
  try
    // Enable parametrized templates
    Template.AllowTagParams := True;
```

```
    // Set template content
    Template.Template := '' + LineEnding +
                         '';

    // Assign tag replacement handler
    Template.OnReplaceTag := @Handler.ReplaceTagHandler;

    // Get parsed content
    Result := Template.GetContent;

    WriteLn(Result);
  finally
    Handler.Free;
    Template.Free;
  end;
end.
```

This will output:

```
<p>Visit our <a href="http://www.freepascal.org">website</a> for more information.</
<p>Logo: <img src="logo.png" alt="FreePascal Logo" /></p>
```

Note that when using parametrized templates:

- Set `AllowTagParams` to `True`
- Use `OnReplaceTag` event instead of `OnGetParam`
- Parameters use the format `[-paramname=value-]`
- The `Values` property is not used in parametrized mode

See also: TTemplateParser.AllowTagParams (1112), TTemplateParser.OnReplaceTag (1110), TReplaceTagEvent (1098)

31.4 Basic usage of the template parser

`TTemplateParser` allows you to replace variables in template text with actual values. Variables are enclosed in delimiters (by default `{` and `}`).

The following example demonstrates basic template parsing with simple variables:

```
program BasicTemplateExample;

{$mode objfpc}{$h+}

uses
  fpTemplate, SysUtils;

var
  Parser: TTemplateParser;
  Template, Result: string;
begin
```

```
  Parser := TTemplateParser.Create;
  try
    // Set up template values
    Parser.Values['Name'] := 'John Doe';
    Parser.Values['Age'] := '30';
    Parser.Values['City'] := 'Amsterdam';

    // Define template text
    Template := 'Hello {Name}! You are {Age} years old and live in {City}.';

    // Parse the template
    Result := Parser.ParseString(Template);

    WriteLn(Result);
  finally
    Parser.Free;
  end;
end.
```

This will output:

```
Hello John Doe! You are 30 years old and live in Amsterdam.
```

You can also use an event handler to provide values dynamically:

```
program EventTemplateExample;

{$mode objfpc}{$h+}

uses
  fpTemplate, SysUtils;

type
  TEventHandler = class
    procedure GetParamValue(Sender: TObject; const ParamName: string; out AValue: st
  end;

procedure TEventHandler.GetParamValue(Sender: TObject; const ParamName: string; out
begin
  case ParamName of
    'CurrentDate': AValue := DateToStr(Now);
    'CurrentTime': AValue := TimeToStr(Now);
    'UserName': AValue := GetEnvironmentVariable('USER');
  else
    AValue := 'Unknown';
  end;
end;

var
  Parser: TTemplateParser;
  Handler: TEventHandler;
  Template, Result: string;
begin
```

```
  Parser := TTemplateParser.Create;
  Handler := TEventHandler.Create;
  try
    // Assign event handler
    Parser.OnGetParam := @Handler.GetParamValue;

    // Define template text
    Template := 'Date: {CurrentDate}, Time: {CurrentTime}, User: {UserName}';

    // Parse the template
    Result := Parser.ParseString(Template);

    WriteLn(Result);
  finally
    Handler.Free;
    Parser.Free;
  end;
end.
```

The `TFPTemplate` class provides a higher-level interface that can load templates from files:

```
program FileTemplateExample;

{$mode objfpc}{$h+}

uses
  fpTemplate, SysUtils;

type
  TEventHandler = class
    procedure GetParamValue(Sender: TObject; const ParamName: string; out AValue: st
  end;

procedure TEventHandler.GetParamValue(Sender: TObject; const ParamName: string; out
begin
  case ParamName of
    'Title': AValue := 'Welcome Page';
    'Content': AValue := 'This is the main content of the page.';
    'Footer': AValue := 'Copyright 2024 Example Company';
  else
    AValue := '';
  end;
end;

var
  Template: TFPTemplate;
  Handler: TEventHandler;
  Result: string;
begin
  Template := TFPTemplate.Create;
  Handler := TEventHandler.Create;
  try
    // Set template content directly
    Template.Template := '';
```

```
    // Assign event handler
    Template.OnGetParam := @Handler.GetParamValue;

    // Get parsed content
    Result := Template.GetContent;

    WriteLn(Result);
  finally
    Handler.Free;
    Template.Free;
  end;
end.
```

See also: TTemplateParser (1106), TFPTemplate (1103), TFPCustomTemplate (1099)

31.5 Constants, types and variables

31.5.1 Constants

```
DefaultParseDepth = 100
```

DefaultParseDepth is the default value for the maximum recursion depth when parsing templates.

```
MaxDelimLength = 5
```

MaxDelimLength is the maximum length (in characters) for a delimiter string.

31.5.2 Types

```
TGetParamEvent = procedure(Sender: TObject; const ParamName: string
  ;
                           out AValue: string) of object
```

TGetParamEvent is the type of the event that is called when the template parser needs the value for a certain template variable.

```
TParseDelimiter = string
```

TParseDelimiter is the type used for the delimiter strings. Its maximum length is MaxDelimLength (1098).

```
TReplaceTagEvent = procedure(Sender: TObject; const TagString: string
  ;
                             TagParams: TStringList;
  out ReplaceText: string) of object
```

TReplaceTagEvent is the type of the event that is called when the template parser needs to replace a tag.

31.5.3 Variables

```
DefaultEndDelimiter : TParseDelimiter = '}'
```

`DefaultEndDelimiter` is the default character which indicates the end of a template variable. By default this is the `}` character.

```
DefaultParamEndDelimiter : TParseDelimiter = '-]'
```

`DefaultParamEndDelimiter` is the default string which indicates the end of a template variable's parameter. By default this is the `-]` string.

```
DefaultParamStartDelimiter : TParseDelimiter = '[-'
```

`DefaultParamStartDelimiter` is the default string which indicates the start of a template variable's parameter. By default this is the `[-` string.

```
DefaultParamValueSeparator : TParseDelimiter = '='
```

`DefaultParamValueSeparator` is the default character which separates a template variable's parameter and its value. By default this is the = character.

```
DefaultStartDelimiter : TParseDelimiter = '{'
```

`DefaultStartDelimiter` is the default character which indicates the start of a template variable. By default this is the `{` character.

```
MaxParseDepth : Integer = DefaultParseDepth
```

`MaxParseDepth` is the maximum recursion depth when parsing templates. If this depth is exceeded, an ETemplateParser (1099) exception is raised.

31.6 ETemplateParser

31.6.1 Description

`ETemplateParser` is the exception class for errors that occur during template parsing.

31.7 TFPCustomTemplate

31.7.1 Description

`TFPCustomTemplate` is a base class for template processing. It can load a template from a file or from a string and then uses a TTemplateParser (1106) instance to parse the template.

Do not use this class directly, use TFPTemplate (1103) instead.

See also: TFPTemplate (1103)

31.7.2 Method overview

Page	Method	Description
1100	Assign	Assign one template to another.
1100	GetContent	Get the parsed template content.
1100	HasContent	Check if the template has content.

31.7.3 Property overview

Page	Properties	Access	Description
1103	AllowTagParams	rw	Allow tags with parameters.
1101	EndDelimiter	rw	End delimiter for a template variable.
1102	FileName	rw	Name of the file with the template.
1102	OnGetParam	rw	Event to get a value for a template variable.
1103	OnReplaceTag	rw	Event to replace a tag.
1101	ParamEndDelimiter	rw	End delimiter for a template variable parameter.
1101	ParamStartDelimiter	rw	Start delimiter for a template variable parameter.
1102	ParamValueSeparator	rw	Separator for a template variable parameter and its value.
1101	StartDelimiter	rw	Start delimiter for a template variable.
1102	Template	rw	The template text.

31.7.4 TFPCustomTemplate.HasContent

Synopsis: Check if the template has content.

Declaration: `function HasContent : Boolean`

Visibility: public

Description: `HasContent` returns `True` if the template has content, i.e. either the FileName (1091) or the Template (1091) property is set.

See also: FileName (1091), Template (1091)

31.7.5 TFPCustomTemplate.GetContent

Synopsis: Get the parsed template content.

Declaration: `function GetContent : string`

Visibility: public

Description: `GetContent` will parse the template and return the result. If the FileName (1091) property is set, the content of the file is parsed. Otherwise the Template (1091) property is parsed.

See also: FileName (1091), Template (1091)

31.7.6 TFPCustomTemplate.Assign

Synopsis: Assign one template to another.

Declaration: `procedure Assign(Source: TPersistent);  Override`

Visibility: public

Description: `Assign` will copy the properties of the `Source` template to the current one.

31.7.7 TFPCustomTemplate.StartDelimiter

Synopsis: Start delimiter for a template variable.

Declaration: `Property StartDelimiter : TParseDelimiter`

Visibility: public

Access: Read,Write

Description: `StartDelimiter` is the character which indicates the start of a template variable. By default this is the `{` character.

See also: DefaultStartDelimiter (1099), EndDelimiter (1091)

31.7.8 TFPCustomTemplate.EndDelimiter

Synopsis: End delimiter for a template variable.

Declaration: `Property EndDelimiter : TParseDelimiter`

Visibility: public

Access: Read,Write

Description: `EndDelimiter` is the character which indicates the end of a template variable. By default this is the `}` character.

See also: DefaultEndDelimiter (1099), StartDelimiter (1091)

31.7.9 TFPCustomTemplate.ParamStartDelimiter

Synopsis: Start delimiter for a template variable parameter.

Declaration: `Property ParamStartDelimiter : TParseDelimiter`

Visibility: public

Access: Read,Write

Description: `ParamStartDelimiter` is the string which indicates the start of a template variable's parameter. By default this is the `[-` string.

See also: DefaultParamStartDelimiter (1099), ParamEndDelimiter (1091)

31.7.10 TFPCustomTemplate.ParamEndDelimiter

Synopsis: End delimiter for a template variable parameter.

Declaration: `Property ParamEndDelimiter : TParseDelimiter`

Visibility: public

Access: Read,Write

Description: `ParamEndDelimiter` is the string which indicates the end of a template variable's parameter. By default this is the `-]` string.

See also: DefaultParamEndDelimiter (1099), ParamStartDelimiter (1091)

31.7.11 TFPCustomTemplate.ParamValueSeparator

Synopsis: Separator for a template variable parameter and its value.

Declaration: `Property ParamValueSeparator : TParseDelimiter`

Visibility: public

Access: Read,Write

Description: `ParamValueSeparator` is the character which separates a template variable's parameter and its value. By default this is the = character.

See also: DefaultParamValueSeparator (1099)

31.7.12 TFPCustomTemplate.FileName

Synopsis: Name of the file with the template.

Declaration: `Property FileName : string`

Visibility: public

Access: Read,Write

Description: `FileName` is the name of the file that contains the template text.

See also: Template (1091)

31.7.13 TFPCustomTemplate.Template

Synopsis: The template text.

Declaration: `Property Template : string`

Visibility: public

Access: Read,Write

Description: `Template` contains the template text.

See also: FileName (1091)

31.7.14 TFPCustomTemplate.OnGetParam

Synopsis: Event to get a value for a template variable.

Declaration: `Property OnGetParam : TGetParamEvent`

Visibility: public

Access: Read,Write

Description: `OnGetParam` is the event that is called when the template parser needs the value for a certain template variable. This event is only used when simple template tags are allowed.

See also: TGetParamEvent (1098), AllowTagParams (1091)

31.7.15 TFPCustomTemplate.OnReplaceTag

Synopsis: Event to replace a tag.

Declaration: `Property OnReplaceTag : TReplaceTagEvent`

Visibility: public

Access: Read,Write

Description: `OnReplaceTag` is the event that is called when the template parser needs to replace a tag. This event is used when template tags with parameters are allowed.

See also: TReplaceTagEvent (1098), AllowTagParams (1091)

31.7.16 TFPCustomTemplate.AllowTagParams

Synopsis: Allow tags with parameters.

Declaration: `Property AllowTagParams : Boolean`

Visibility: public

Access: Read,Write

Description: If `AllowTagParams` is set to `True`, then tags with parameters are allowed. If `False`, then only simple tags are allowed.

31.8 TFPTemplate

31.8.1 Description

`TFPTemplate` is a component that can be used to parse templates. It is a descendant of TFPCustomTemplate (1099) and publishes the properties of its ancestor.

It can be dropped on a form and its properties can be set in the object inspector.

See also: TFPCustomTemplate (1099)

31.8.2 Property overview

Page	Properties	Access	Description
1104	AllowTagParams		Allow tags with parameters.
1105	EndDelimiter		End delimiter for a template variable.
1104	FileName		Name of the file with the template.
1106	OnGetParam		Event to get a value for a template variable.
1104	OnReplaceTag		Event to replace a tag.
1105	ParamEndDelimiter		End delimiter for a template variable parameter.
1105	ParamStartDelimiter		Start delimiter for a template variable parameter.
1106	ParamValueSeparator		Separator for a template variable parameter and its value.
1105	StartDelimiter		Start delimiter for a template variable.
1104	Template		The template text.

31.8.3 TFPTemplate.FileName

Synopsis: Name of the file with the template.

Declaration: `Property FileName :`

Visibility: published

Access:

Description: See TFPCustomTemplate.FileName (1102).

See also: TFPCustomTemplate.FileName (1102)

31.8.4 TFPTemplate.Template

Synopsis: The template text.

Declaration: `Property Template :`

Visibility: published

Access:

Description: See TFPCustomTemplate.Template (1102).

See also: TFPCustomTemplate.Template (1102)

31.8.5 TFPTemplate.AllowTagParams

Synopsis: Allow tags with parameters.

Declaration: `Property AllowTagParams :`

Visibility: published

Access:

Description: See TFPCustomTemplate.AllowTagParams (1103).

See also: TFPCustomTemplate.AllowTagParams (1103)

31.8.6 TFPTemplate.OnReplaceTag

Synopsis: Event to replace a tag.

Declaration: `Property OnReplaceTag :`

Visibility: published

Access:

Description: See TFPCustomTemplate.OnReplaceTag (1103).

See also: TFPCustomTemplate.OnReplaceTag (1103)

31.8.7 TFPTemplate.StartDelimiter

Synopsis: Start delimiter for a template variable.

Declaration: `Property StartDelimiter :`

Visibility: published

Access:

Description: See TFPCustomTemplate.StartDelimiter (1101).

See also: TFPCustomTemplate.StartDelimiter (1101)

31.8.8 TFPTemplate.EndDelimiter

Synopsis: End delimiter for a template variable.

Declaration: `Property EndDelimiter :`

Visibility: published

Access:

Description: See TFPCustomTemplate.EndDelimiter (1101).

See also: TFPCustomTemplate.EndDelimiter (1101)

31.8.9 TFPTemplate.ParamStartDelimiter

Synopsis: Start delimiter for a template variable parameter.

Declaration: `Property ParamStartDelimiter :`

Visibility: published

Access:

Description: See TFPCustomTemplate.ParamStartDelimiter (1101).

See also: TFPCustomTemplate.ParamStartDelimiter (1101)

31.8.10 TFPTemplate.ParamEndDelimiter

Synopsis: End delimiter for a template variable parameter.

Declaration: `Property ParamEndDelimiter :`

Visibility: published

Access:

Description: See TFPCustomTemplate.ParamEndDelimiter (1101).

See also: TFPCustomTemplate.ParamEndDelimiter (1101)

31.8.11 TFPTemplate.ParamValueSeparator

Synopsis: Separator for a template variable parameter and its value.

Declaration: `Property ParamValueSeparator :`

Visibility: published

Access:

Description: See TFPCustomTemplate.ParamValueSeparator (1102).

See also: TFPCustomTemplate.ParamValueSeparator (1102)

31.8.12 TFPTemplate.OnGetParam

Synopsis: Event to get a value for a template variable.

Declaration: `Property OnGetParam :`

Visibility: published

Access:

Description: See TFPCustomTemplate.OnGetParam (1102).

See also: TFPCustomTemplate.OnGetParam (1102)

31.9 TTemplateParser

31.9.1 Description

`TTemplateParser` is a class that can parse a text and substitute template variables with their values.

The template variables are enclosed in delimiters, by default `{` and `}`. Whenever a variable is found, its value is looked up in the Values (1091) property. If it is not found there, the OnGetParam (1091) event is called to obtain a value for the variable.

The delimiters can be set using the various delimiter properties.

See also: TFPCustomTemplate (1099)

31.9.2 Method overview

Page	Method	Description
1107	Clear	Clear the list of values.
1107	Create	Create a new instance of the template parser.
1107	Destroy	Frees the template parser instance.
1108	GetParam	Get a value for a parameter.
1108	GetTagParams	Extract tag parameters from a tag name.
1109	ParseFiles	Parse a file and substitute variables.
1109	ParseStream	Parse a stream and substitute variables.
1108	ParseString	Parse a string and substitute variables.
1109	ParseStrings	Parse a stringlist and substitute variables.
1108	ReplaceTag	Replace a tag with a value.

31.9.3 Property overview

Page	Properties	Access	Description
1112	AllowTagParams	rw	Allow tags with parameters.
1110	EndDelimiter	rw	End delimiter for a template variable.
1112	NamesByIndex	r	Access to variable names by index.
1109	OnGetParam	rw	Event to get a value for a template variable.
1110	OnReplaceTag	rw	Event to replace a tag.
1111	ParamEndDelimiter	rw	End delimiter for a template variable parameter.
1110	ParamStartDelimiter	rw	Start delimiter for a template variable parameter.
1111	ParamValueSeparator	rw	Separator for a template variable parameter and its value.
1112	Recursive	rw	Enable recursive parsing.
1110	StartDelimiter	rw	Start delimiter for a template variable.
1112	ValueCount	r	Number of values in the list.
1111	Values	rw	List of values for template variables.
1111	ValuesByIndex	r	Access to values by index.

31.9.4 TTemplateParser.Create

Synopsis: Create a new instance of the template parser.

Declaration: `constructor Create`

Visibility: public

Description: `Create` instantiates a new `TTemplateParser` object. It initializes the delimiters to their default values and sets the maximum parse depth.

See also: Destroy (1091)

31.9.5 TTemplateParser.Destroy

Synopsis: Frees the template parser instance.

Declaration: `destructor Destroy;  Override`

Visibility: public

Description: `Destroy` cleans up the `TTemplateParser` instance and frees any memory allocated by it. In particular, it will free the list of values.

See also: Create (1091)

31.9.6 TTemplateParser.Clear

Synopsis: Clear the list of values.

Declaration: `procedure Clear`

Visibility: public

Description: `Clear` removes all values from the list of values.

See also: Values (1091)

31.9.7 TTemplateParser.ReplaceTag

Synopsis: Replace a tag with a value.

Declaration:
```
function ReplaceTag(const Key: string; TagParams: TStringList;
                    out ReplaceWith: string) : Boolean
```

Visibility: public

Description: `ReplaceTag` will replace the tag with name `Key` and parameters `TagParams` with the value in `ReplaceWith`.

The function returns `True` if the tag was replaced, `False` otherwise. A replacement will happen if the OnReplaceTag (1091) event is assigned.

See also: OnReplaceTag (1091)

31.9.8 TTemplateParser.GetParam

Synopsis: Get a value for a parameter.

Declaration:
```
function GetParam(const Key: string; out AValue: string) : Boolean
```

Visibility: public

Description: `GetParam` will get the value for the parameter with name `Key` and return it in `AValue`.

First, the Values (1091) list is consulted. If the parameter is not found there, the OnGetParam (1091) event is called. If the parameter is found, the function returns `True`, otherwise it returns `False`.

See also: Values (1091), OnGetParam (1091)

31.9.9 TTemplateParser.GetTagParams

Synopsis: Extract tag parameters from a tag name.

Declaration:
```
procedure GetTagParams(var TagName: string; var TagParams: TStringList)
```

Visibility: public

Description: `GetTagParams` will extract any tag parameters from the tag name `TagName` and return them in the `TagParams` stringlist. The tag name itself is also adapted.

31.9.10 TTemplateParser.ParseString

Synopsis: Parse a string and substitute variables.

Declaration:
```
function ParseString(const Src: string) : string
```

Visibility: public

Description: `ParseString` will parse the string `Src` and substitute all template variables with their values. The resulting string is returned.

See also: ParseStream (1091), ParseStrings (1091), ParseFiles (1091)

31.9.11 TTemplateParser.ParseStream

Synopsis: Parse a stream and substitute variables.

Declaration: `function ParseStream(Src: TStream; Dest: TStream) : Integer`

Visibility: public

Description: `ParseStream` will read the content of stream `Src`, substitute all template variables with their values and write the result to stream `Dest`. The function returns the number of bytes written to the destination stream.

See also: ParseString (1091), ParseStrings (1091), ParseFiles (1091)

31.9.12 TTemplateParser.ParseStrings

Synopsis: Parse a stringlist and substitute variables.

Declaration: `procedure ParseStrings(Src: TStrings; Dest: TStrings)`

Visibility: public

Description: `ParseStrings` will read the strings from the stringlist `Src`, substitute all template variables with their values and add the resulting strings to the stringlist `Dest`.

See also: ParseString (1091), ParseStream (1091), ParseFiles (1091)

31.9.13 TTemplateParser.ParseFiles

Synopsis: Parse a file and substitute variables.

Declaration: `procedure ParseFiles(const Src: string; const Dest: string)`

Visibility: public

Description: `ParseFiles` will read the content of file `Src`, substitute all template variables with their values and write the result to file `Dest`.

See also: ParseString (1091), ParseStream (1091), ParseStrings (1091)

31.9.14 TTemplateParser.OnGetParam

Synopsis: Event to get a value for a template variable.

Declaration: `Property OnGetParam : TGetParamEvent`

Visibility: public

Access: Read,Write

Description: `OnGetParam` is the event that is called when the template parser needs the value for a certain template variable. This event is only used when simple template tags are allowed.

See also: TGetParamEvent (1098), AllowTagParams (1091)

31.9.15 TTemplateParser.OnReplaceTag

Synopsis: Event to replace a tag.

Declaration: `Property OnReplaceTag : TReplaceTagEvent`

Visibility: public

Access: Read,Write

Description: `OnReplaceTag` is the event that is called when the template parser needs to replace a tag. This event is used when template tags with parameters are allowed.

See also: TReplaceTagEvent (1098), AllowTagParams (1091)

31.9.16 TTemplateParser.StartDelimiter

Synopsis: Start delimiter for a template variable.

Declaration: `Property StartDelimiter : TParseDelimiter`

Visibility: public

Access: Read,Write

Description: `StartDelimiter` is the character which indicates the start of a template variable. By default this is the `{` character.

See also: DefaultStartDelimiter (1099), EndDelimiter (1091)

31.9.17 TTemplateParser.EndDelimiter

Synopsis: End delimiter for a template variable.

Declaration: `Property EndDelimiter : TParseDelimiter`

Visibility: public

Access: Read,Write

Description: `EndDelimiter` is the character which indicates the end of a template variable. By default this is the `}` character.

See also: DefaultEndDelimiter (1099), StartDelimiter (1091)

31.9.18 TTemplateParser.ParamStartDelimiter

Synopsis: Start delimiter for a template variable parameter.

Declaration: `Property ParamStartDelimiter : TParseDelimiter`

Visibility: public

Access: Read,Write

Description: `ParamStartDelimiter` is the string which indicates the start of a template variable's parameter. By default this is the `[-` string.

See also: DefaultParamStartDelimiter (1099), ParamEndDelimiter (1091)

31.9.19 TTemplateParser.ParamEndDelimiter

Synopsis: End delimiter for a template variable parameter.

Declaration: `Property ParamEndDelimiter : TParseDelimiter`

Visibility: public

Access: Read,Write

Description: `ParamEndDelimiter` is the string which indicates the end of a template variable's parameter. By default this is the `-]` string.

See also: DefaultParamEndDelimiter (1099), ParamStartDelimiter (1091)

31.9.20 TTemplateParser.ParamValueSeparator

Synopsis: Separator for a template variable parameter and its value.

Declaration: `Property ParamValueSeparator : TParseDelimiter`

Visibility: public

Access: Read,Write

Description: `ParamValueSeparator` is the character which separates a template variable's parameter and its value. By default this is the = character.

See also: DefaultParamValueSeparator (1099)

31.9.21 TTemplateParser.Values

Synopsis: List of values for template variables.

Declaration: `Property Values[Key: string]: string`

Visibility: public

Access: Read,Write

Description: `Values` is a stringlist that contains key-value pairs. The key is the name of the template variable, and the value is the string to replace the variable with. This property is only used when simple template tags are allowed.

See also: OnGetParam (1091), AllowTagParams (1091)

31.9.22 TTemplateParser.ValuesByIndex

Synopsis: Access to values by index.

Declaration: `Property ValuesByIndex[index: Integer]: string`

Visibility: public

Access: Read

Description: `ValuesByIndex` provides access to the values in the Values (1091) list by index. This property is only used when simple template tags are allowed.

See also: Values (1091), AllowTagParams (1091)

31.9.23 TTemplateParser.NamesByIndex

Synopsis: Access to variable names by index.

Declaration: `Property NamesByIndex[index: Integer]: string`

Visibility: public

Access: Read

Description: `NamesByIndex` provides access to the names of the variables in the Values (1091) list by index. This property is only used when simple template tags are allowed.

See also: Values (1091), AllowTagParams (1091)

31.9.24 TTemplateParser.ValueCount

Synopsis: Number of values in the list.

Declaration: `Property ValueCount : Integer`

Visibility: public

Access: Read

Description: `ValueCount` is the number of values in the Values (1091) list. This property is only used when simple template tags are allowed.

See also: Values (1091), AllowTagParams (1091)

31.9.25 TTemplateParser.Recursive

Synopsis: Enable recursive parsing.

Declaration: `Property Recursive : Boolean`

Visibility: public

Access: Read,Write

Description: If `Recursive` is set to `True`, then the substituted values are parsed again for template variables. This property is only used when simple template tags are allowed.

See also: AllowTagParams (1091)

31.9.26 TTemplateParser.AllowTagParams

Synopsis: Allow tags with parameters.

Declaration: `Property AllowTagParams : Boolean`

Visibility: public

Access: Read,Write

Description: If `AllowTagParams` is set to `True`, then tags with parameters are allowed. If `False`, then only simple tags are allowed.

Chapter 32

Reference for unit 'fptimer'

32.1 Used units

Table 32.1: Used units by unit 'fptimer'

Name	Page
Classes	??
System	??

32.2 Overview

The fpTimer unit implements a timer class TFPTimer (1115) which can be used on all supported platforms. The timer class uses a driver class TFPTimerDriver (1117) which does the actual work.

A default timer driver class is implemented on all platforms. It will work in GUI and non-gui applications, but only in the application's main thread.

An alternative driver class can be used by setting the DefaultTimerDriverClass (1113) variable to the class pointer of the driver class. The driver class should descend from TFPTimerDriver (1117).

32.3 Constants, types and variables

32.3.1 Types

```
TFPTimerDriverClass = Class of TFPTimerDriver
```

`TFPTimerDriverClass` is the class pointer of TFPTimerDriver (1117) it exists mainly for the purpose of being able to set DefaultTimerDriverClass (1113), so a custom timer driver can be used for the timer instances.

32.3.2 Variables

```
DefaultTimerDriverClass : TFPTimerDriverClass = Nil
```

DefaultTimerDriverClass contains the TFPTimerDriver (1117) class pointer that should be used when a new instance of TFPCustomTimer (1114) is created. It is by default set to the system timer class.

Setting this class pointer to another descendent of TFPTimerDriver allows to customize the default timer implementation used in the entire application.

32.4 TFPCustomTimer

32.4.1 Description

TFPCustomTimer is the timer class containing the timer's implementation. It relies on an extra driver instance (of type TFPTimerDriver (1117)) to do the actual work.

TFPCustomTimer publishes no events or properties, so it is unsuitable for handling in an IDE. The TFPTimer (1115) descendent class publishes all needed events of TFPCustomTimer.

See also: TFPTimerDriver (1117), TFPTimer (1115)

32.4.2 Method overview

Page	Method	Description
1114	Create	Create a new timer.
1114	Destroy	Release a timer instance from memory.
1115	StartTimer	Start the timer.
1115	StopTimer	Stop the timer.

32.4.3 TFPCustomTimer.Create

Synopsis: Create a new timer.

Declaration: `constructor Create(AOwner: TComponent); Override`

Visibility: public

Description: Create instantiates a new TFPCustomTimer instance. It creates the timer driver instance from the DefaultTimerDriverClass class pointer.

See also: TFPCustomTimer.Destroy (1114)

32.4.4 TFPCustomTimer.Destroy

Synopsis: Release a timer instance from memory.

Declaration: `destructor Destroy; Override`

Visibility: public

Description: Destroy releases the timer driver component from memory, and then calls Inherited to clean the TFPCustomTimer instance from memory.

See also: TFPCustomTimer.Create (1114)

32.4.5 TFPCustomTimer.StartTimer

Synopsis: Start the timer.

Declaration: `procedure StartTimer;  Virtual`

Visibility: public

Description: `StartTimer` starts the timer. After a call to `StartTimer`, the timer will start producing timer ticks.

The timer stops producing ticks only when the StopTimer (1115) event is called.

See also: StopTimer (1115), Enabled (1115), OnTimer (1116)

32.4.6 TFPCustomTimer.StopTimer

Synopsis: Stop the timer.

Declaration: `procedure StopTimer;  Virtual`

Visibility: public

Description: `Stoptimer` stops a started timer. After a call to `StopTimer`, the timer no longer produces timer ticks.

See also: StartTimer (1115), Enabled (1115), OnTimer (1116)

32.5 TFPTimer

32.5.1 Description

`TFPTimer` implements no new events or properties, but merely publishes events and properties already implemented in TFPCustomTimer (1114): Enabled (1115), OnTimer (1116) and Interval (1116).

The `TFPTimer` class is suitable for use in an IDE.

See also: TFPCustomTimer (1114), Enabled (1115), OnTimer (1116), Interval (1116)

32.5.2 Property overview

Page	Properties	Access	Description
1115	Enabled		Start or stop the timer.
1116	Interval		Timer tick interval in milliseconds.
1116	OnStartTimer		
1117	OnStopTimer		
1116	OnTimer		Event called on each timer tick.
1116	UseTimerThread		

32.5.3 TFPTimer.Enabled

Synopsis: Start or stop the timer.

Declaration: `Property Enabled :`

Visibility: published

Access:

Description: `Enabled` controls whether the timer is active. Setting `Enabled` to `True` will start the timer (calling StartTimer (1115)), setting it to `False` will stop the timer (calling StopTimer (1115)).

See also: StartTimer (1115), StopTimer (1115), OnTimer (1116), Interval (1116)

32.5.4 TFPTimer.Interval

Synopsis: Timer tick interval in milliseconds.

Declaration: `Property Interval :`

Visibility: published

Access:

Description: `Interval` specifies the timer interval in milliseconds. Every `Interval` milliseconds, the OnTimer (1116) event handler will be called.

Note that the milliseconds interval is a minimum interval. Under high system load, the timer tick may arrive later.

See also: OnTimer (1116), Enabled (1115)

32.5.5 TFPTimer.UseTimerThread

Declaration: `Property UseTimerThread :`

Visibility: published

Access:

32.5.6 TFPTimer.OnTimer

Synopsis: Event called on each timer tick.

Declaration: `Property OnTimer :`

Visibility: published

Access:

Description: `OnTimer` is called on each timer tick. The event handler must be assigned to a method that will do the actual work that should occur when the timer fires.

See also: Interval (1116), Enabled (1115)

32.5.7 TFPTimer.OnStartTimer

Declaration: `Property OnStartTimer :`

Visibility: published

Access:

32.5.8 TFPTimer.OnStopTimer

Declaration: `Property OnStopTimer :`

Visibility: published

Access:

32.6 TFPTimerDriver

32.6.1 Description

`TFPTimerDriver` is the abstract timer driver class: it simply provides an interface for the TFPCustomTimer (1114) class to use.

The `fpTimer` unit implements a descendent of this class which implements the default timer mechanism.

See also: TFPCustomTimer (1114), DefaultTimerDriverClass (1113)

32.6.2 Method overview

Page	Method	Description
1117	Create	Creates a new driver instance.
1117	StartTimer	Start the timer.
1118	StopTimer	Stop the timer.

32.6.3 Property overview

Page	Properties	Access	Description
1118	Timer	r	Timer tick.
1118	TimerStarted	r	True when the timer driver has called its StartTimer method.

32.6.4 TFPTimerDriver.Create

Synopsis: Creates a new driver instance.

Declaration: `constructor Create(ATimer: TFPCustomTimer); Virtual`

Visibility: public

Description: `Create` should be overridden by descendents of `TFPTimerDriver` to do additional initialization of the timer driver. `Create` just stores (in Timer (1118)) a reference to the `ATimer` instance which created the driver instance.

See also: Timer (1118), TFPTimer (1115)

32.6.5 TFPTimerDriver.StartTimer

Synopsis: Start the timer.

Declaration: `procedure StartTimer; Virtual; Abstract`

Visibility: public

Description: `StartTimer` is called by TFPCustomTimer.StartTimer (1115). It should be overridden by descendents of `TFPTimerDriver` to actually start the timer.

See also: TFPCustomTimer.StartTimer (1115), TFPTimerDriver.StopTimer (1118)

32.6.6 TFPTimerDriver.StopTimer

Synopsis: Stop the timer.

Declaration: `procedure StopTimer; Virtual; Abstract`

Visibility: public

Description: `StopTimer` is called by TFPCustomTimer.StopTimer (1115). It should be overridden by descendents of `TFPTimerDriver` to actually stop the timer.

See also: TFPCustomTimer.StopTimer (1115), TFPTimerDriver.StartTimer (1117)

32.6.7 TFPTimerDriver.Timer

Synopsis: Timer tick.

Declaration: `Property Timer : TFPCustomTimer`

Visibility: public

Access: Read

Description: `Timer` calls the TFPCustomTimer (1114) timer event. Descendents of `TFPTimerDriver` should call `Timer` whenever a timer tick occurs.

See also: TFPTimer.OnTimer (1116), TFPTimerDriver.StartTimer (1117), TFPTimerDriver.StopTimer (1118)

32.6.8 TFPTimerDriver.TimerStarted

Synopsis: True when the timer driver has called its StartTimer method.

Declaration: `Property TimerStarted : Boolean`

Visibility: public

Access: Read

Description: True when the timer driver has called its StartTimer method.

Chapter 33

Reference for unit 'fpWeb'

33.1 Used units

Table 33.1: Used units by unit 'fpWeb'

Name	Page
Classes	??
fphttp	778
fpTemplate	1091
HTTPDefs	1141
System	??
sysutils	??

33.2 Overview

The `fpWeb` unit provides HTML-oriented functionality for web application development. It extends the basic HTTP functionality with support for web actions, template processing, and structured web modules.

This unit introduces the concept of web actions which allow mapping of URL patterns to specific handlers. It also provides template support for dynamic content generation and a complete web module framework for building complex web applications.

33.3 Constants, types and variables

33.3.1 Resource strings

```
SErrInvalidVar = 'Invalid template variable name : "%s"'
```

Resource string used when an invalid template variable name is encountered.

```
SErrInvalidWebAction = 'Invalid action for "%s".'
```

Resource string used when an invalid web action is encountered during request processing.

```
SErrNoContentProduced = 'No template content was produced.'
```

Resource string used when no content is produced during template or action processing.

33.3.2 Types

```
TContentEvent = procedure(Sender: TObject; Content: TStream) of
  object
```

Event type used for content generation callbacks. The event passes a content stream that can be written to.

33.4 EFPWebError

33.4.1 Description

`EFPWebError` is the exception class used throughout the fpweb unit for web-related errors. It extends EHTTP (1119) to provide specific error handling for web application functionality.

See also: EHTTP (1119)

33.5 TCustomFPWebModule

33.5.1 Description

`TCustomFPWebModule` extends TSessionHTTPModule (1119) with support for web actions, template processing, and advanced request handling. This class provides the foundation for building complex web applications with structured request routing and dynamic content generation.

The module manages a collection of web actions, template variables, and provides hooks for request preprocessing and response postprocessing.

See also: TFPWebModule (1129), TFPWebActions (1127), TSessionHTTPModule (1119)

33.5.2 Method overview

Page	Method	Description
1121	CreateNew	Creates a new web module instance
1121	Destroy	Destroys the web module instance
1121	HandleRequest	Processes an incoming HTTP request

33.5.3 Property overview

Page	Properties	Access	Description
1122	Actions	rw	Collection of web actions for this module
1122	ActionVar	rw	Name of the request parameter used to identify actions
1123	AfterResponse	rw	Event called after response processing
1122	BeforeRequest	rw	Event called before request processing
1123	DefActionWhenUnknown	rw	Whether to use the default action when no specific action is found
1124	ModuleTemplate	rw	Template used for content generation
1123	OnGetAction	rw	Event called to determine which action to execute
1124	OnGetParam	rw	Event called to resolve template parameters
1123	OnRequest	rw	Event called for request processing
1124	OnTemplateContent	rw	Event called for template content processing
1124	Request	r	The current HTTP request being processed
1125	Response	r	The current HTTP response being generated

33.5.4 TCustomFPWebModule.CreateNew

Synopsis: Creates a new web module instance

Declaration:
```
constructor CreateNew(AOwner: TComponent; CreateMode: Integer)
                      ;  Override
```

Visibility: public

Description: Creates a new web module and initializes the actions collection, template, and template variables. This constructor is used instead of the standard Create constructor to support the component creation mode.

See also: Destroy (1121)

33.5.5 TCustomFPWebModule.Destroy

Synopsis: Destroys the web module instance

Declaration:
```
destructor Destroy;  Override
```

Visibility: public

Description: Destroys the web module and frees all associated resources including template variables, template, and actions collection.

See also: CreateNew (1121)

33.5.6 TCustomFPWebModule.HandleRequest

Synopsis: Processes an incoming HTTP request

Declaration:
```
procedure HandleRequest(ARequest: TRequest; AResponse: TResponse)
                        ;  Override
```

Visibility: public

Description: Main request handling method that processes incoming HTTP requests. This method coordinates session management, action execution, template processing, and response generation.

The method calls various hooks in sequence: `BeforeRequest`, `OnRequest`, action processing, template processing, and `AfterResponse`.

Errors: May raise EFPWebError (1120) if the request cannot be handled.

See also: BeforeRequest (1122), OnRequest (1123), AfterResponse (1123)

33.5.7 TCustomFPWebModule.Actions

Synopsis: Collection of web actions for this module

Declaration: `Property Actions : TFPWebActions`

Visibility: public

Access: Read,Write

Description: Contains the collection of web actions that can be executed by this web module. Actions are selected based on request parameters and executed to handle specific requests.

See also: TFPWebActions (1127), ActionVar (1122)

33.5.8 TCustomFPWebModule.ActionVar

Synopsis: Name of the request parameter used to identify actions

Declaration: `Property ActionVar : string`

Visibility: public

Access: Read,Write

Description: Specifies the name of the request parameter that contains the action name to execute. This is typically a URL parameter like 'action' or 'command'.

See also: Actions (1122)

33.5.9 TCustomFPWebModule.BeforeRequest

Synopsis: Event called before request processing

Declaration: `Property BeforeRequest : TRequestEvent`

Visibility: public

Access: Read,Write

Description: Event triggered before the main request processing begins. This can be used for request preprocessing, authentication, or logging.

See also: TRequestEvent (1119), AfterResponse (1123)

33.5.10 TCustomFPWebModule.OnRequest

Synopsis: Event called for request processing

Declaration: `Property OnRequest : TWebActionEvent`

Visibility: public

Access: Read,Write

Description: Event triggered during request processing. If this event handles the request and sets the Handled parameter to `True`, no further processing occurs.

See also: TWebActionEvent (1119), BeforeRequest (1122)

33.5.11 TCustomFPWebModule.AfterResponse

Synopsis: Event called after response processing

Declaration: `Property AfterResponse : TResponseEvent`

Visibility: public

Access: Read,Write

Description: Event triggered after the response has been generated and sent. This can be used for cleanup, logging, or response postprocessing.

See also: TResponseEvent (1119), BeforeRequest (1122)

33.5.12 TCustomFPWebModule.OnGetAction

Synopsis: Event called to determine which action to execute

Declaration: `Property OnGetAction : TGetActionEvent`

Visibility: public

Access: Read,Write

Description: Event that allows custom logic for determining which action should be executed for a given request. This provides more flexibility than the standard action variable lookup.

See also: TGetActionEvent (1119), ActionVar (1122)

33.5.13 TCustomFPWebModule.DefActionWhenUnknown

Synopsis: Whether to use the default action when no specific action is found

Declaration: `Property DefActionWhenUnknown : Boolean`

Visibility: public

Access: Read,Write

Description: Determines whether the default action should be executed when no action name is specified in the request or when the specified action cannot be found.

See also: Actions (1122)

33.5.14 TCustomFPWebModule.ModuleTemplate

Synopsis: Template used for content generation

Declaration: `Property ModuleTemplate : TFPTemplate`

Visibility: public

Access: Read,Write

Description: Template object used for dynamic content generation when no action handles the request. The template can contain variables that are substituted during processing.

See also: TFPTemplate (1119), OnGetParam (1124)

33.5.15 TCustomFPWebModule.OnGetParam

Synopsis: Event called to resolve template parameters

Declaration: `Property OnGetParam : TGetParamEvent`

Visibility: public

Access: Read,Write

Description: Event triggered when a template parameter needs to be resolved during template processing. This allows dynamic parameter value generation.

See also: TGetParamEvent (1119), ModuleTemplate (1124)

33.5.16 TCustomFPWebModule.OnTemplateContent

Synopsis: Event called for template content processing

Declaration: `Property OnTemplateContent : TGetParamEvent`

Visibility: public

Access: Read,Write

Description: Event triggered for template content processing. This is an alias for the `OnGetParam` event and provides the same functionality.

See also: OnGetParam (1124), TGetParamEvent (1119)

33.5.17 TCustomFPWebModule.Request

Synopsis: The current HTTP request being processed

Declaration: `Property Request : TRequest`

Visibility: public

Access: Read

Description: Provides access to the HTTP request currently being processed. This property is only valid during request processing and is set to `nil` when no request is active.

See also: TRequest (1119), Response (1125)

33.5.18 TCustomFPWebModule.Response

Synopsis: The current HTTP response being generated

Declaration: `Property Response : TResponse`

Visibility: public

Access: Read

Description: Provides access to the HTTP response currently being generated. This property is only valid during request processing and is set to `nil` when no request is active.

See also: TResponse (1119), Request (1124)

33.6 TFPWebAction

33.6.1 Description

`TFPWebAction` extends TCustomWebAction (1119) with support for direct content assignment and template processing. This class allows actions to serve static content from string lists or process dynamic content through templates.

The action can handle requests through the `OnRequest` event and provides built-in content serving capabilities.

See also: TFPWebActions (1127), TCustomWebAction (1119), TFPTemplate (1119)

33.6.2 Method overview

Page	Method	Description
1126	Assign	Assigns properties from another object
1125	create	Creates a new `TFPWebAction` instance
1126	destroy	Destroys the `TFPWebAction` instance

33.6.3 Property overview

Page	Properties	Access	Description
1126	Contents	rw	Static content to serve when the action is executed
1126	OnRequest	rw	Event handler called when this action is executed
1127	Template	rw	Template for dynamic content generation

33.6.4 TFPWebAction.create

Synopsis: Creates a new `TFPWebAction` instance

Declaration: `constructor create(ACollection: TCollection);  Override`

Visibility: public

Description: Creates a new `TFPWebAction` instance and initializes the internal template object. The action is added to the specified collection.

See also: destroy (1126)

33.6.5 TFPWebAction.destroy

Synopsis: Destroys the `TFPWebAction` instance

Declaration: `destructor destroy;  Override`

Visibility: public

Description: Destroys the `TFPWebAction` instance and frees the associated content strings and template objects.

See also: create (1125)

33.6.6 TFPWebAction.Assign

Synopsis: Assigns properties from another object

Declaration: `procedure Assign(Source: TPersistent);  Override`

Visibility: public

Description: Copies all properties from the source `TFPWebAction` object to this instance. This includes name, events, content, and template settings.

33.6.7 TFPWebAction.Contents

Synopsis: Static content to serve when the action is executed

Declaration: `Property Contents : TStrings`

Visibility: published

Access: Read,Write

Description: Contains static text content that will be served when this action is executed. If the action's `OnRequest` event does not handle the request, this content will be returned to the client.

See also: OnRequest (1126), Template (1127)

33.6.8 TFPWebAction.OnRequest

Synopsis: Event handler called when this action is executed

Declaration: `Property OnRequest : TWebActionEvent`

Visibility: published

Access: Read,Write

Description: Event that is triggered when this web action is executed. The event handler receives the request and response objects and can set the Handled parameter to `True` to indicate that the request has been processed.

See also: TWebActionEvent (1119), Contents (1126)

33.6.9 TFPWebAction.Template

Synopsis: Template for dynamic content generation

Declaration: `Property Template : TFPTemplate`

Visibility: published

Access: Read,Write

Description: Template object used for processing dynamic content. The template can contain variable substitutions that are resolved at runtime to generate the final output.

See also: TFPTemplate (1119), Contents (1126)

33.7 TFPWebActions

33.7.1 Description

`TFPWebActions` extends TCustomWebActions (1119) to provide a collection of TFPWebAction (1125) objects. It handles request routing to appropriate actions and provides content generation capabilities.

The collection maintains a reference to the currently executing action and can route requests based on URL parameters.

See also: TFPWebAction (1125), TCustomWebActions (1119)

33.7.2 Method overview

Page	Method	Description
1128	ActionByName	Returns the action with the specified name
1127	Add	Adds a new web action to the collection
1128	FindAction	Searches for an action with the specified name

33.7.3 Property overview

Page	Properties	Access	Description
1128	ActionVar		Name of the request parameter used to identify the action
1129	CurrentAction	r	The currently executing web action
1128	FPWebActions	rw	Array property providing indexed access to web actions

33.7.4 TFPWebActions.Add

Synopsis: Adds a new web action to the collection

Declaration: `function Add : TFPWebAction`

Visibility: public

Description: Creates and adds a new `TFPWebAction` to the collection. The newly created action is returned for further configuration.

See also: TFPWebAction (1125), FindAction (1128)

33.7.5 TFPWebActions.ActionByName

Synopsis: Returns the action with the specified name

Declaration: `function ActionByName(const AName: string) : TFPWebAction`

Visibility: public

Description: Finds and returns the web action with the given name. Raises an exception if no action with the specified name is found.

Errors: Raises an exception if the action name is not found.

See also: FindAction (1128)

33.7.6 TFPWebActions.FindAction

Synopsis: Searches for an action with the specified name

Declaration: `function FindAction(const AName: string) : TFPWebAction`

Visibility: public

Description: Searches for a web action with the given name. Returns `nil` if no action with the specified name is found.

See also: ActionByName (1128)

33.7.7 TFPWebActions.FPWebActions

Synopsis: Array property providing indexed access to web actions

Declaration: `Property FPWebActions[Index: Integer]: TFPWebAction; default`

Visibility: public

Access: Read,Write

Description: Provides indexed access to the web actions in the collection. This is the default array property for the class.

See also: TFPWebAction (1125)

33.7.8 TFPWebActions.ActionVar

Synopsis: Name of the request parameter used to identify the action

Declaration: `Property ActionVar :`

Visibility: public

Access:

Description: Specifies the name of the request parameter (usually a URL parameter) that contains the action name to execute. Common values include 'action', 'command', or 'page'.

See also: FindAction (1128)

33.7.9 TFPWebActions.CurrentAction

Synopsis: The currently executing web action

Declaration: `Property CurrentAction : TFPWebAction`

Visibility: public

Access: Read

Description: References the web action that is currently being executed. This property is set during request processing and is available for the duration of the action execution.

See also: TFPWebAction (1125)

33.8 TFPWebModule

33.8.1 Description

`TFPWebModule` is the complete implementation of a web module that publishes all properties from TCustomFPWebModule (1120). This class is intended for use in visual design environments and provides access to all web module functionality through published properties.

See also: TCustomFPWebModule (1120)

33.8.2 Property overview

Page	Properties	Access	Description
1129	Actions		Collection of web actions for this module
1130	ActionVar		Name of the request parameter used to identify actions
1132	AfterInitModule		Event called after module initialization
1130	AfterResponse		Event called after response processing
1130	BeforeRequest		Event called before request processing
1133	CORS		Cross-Origin Resource Sharing configuration
1131	CreateSession		Whether to automatically create sessions
1131	DefActionWhenUnknown		Whether to use the default action when no specific action is found
1132	Kind		The module kind
1131	OnGetAction		Event called to determine which action to execute
1132	OnNewSession		Event called when a new session is created
1130	OnRequest		Event called for request processing
1132	OnSessionExpired		Event called when a session expires
1131	Session		The current session object

33.8.3 TFPWebModule.Actions

Synopsis: Collection of web actions for this module

Declaration: `Property Actions :`

Visibility: published

Access:

Description: Published version of TCustomFPWebModule.Actions (1122).

See also: TCustomFPWebModule.Actions (1122)

33.8.4 TFPWebModule.ActionVar

Synopsis: Name of the request parameter used to identify actions

Declaration: `Property ActionVar :`

Visibility: published

Access:

Description: Published version of TCustomFPWebModule.ActionVar (1122).

See also: TCustomFPWebModule.ActionVar (1122)

33.8.5 TFPWebModule.BeforeRequest

Synopsis: Event called before request processing

Declaration: `Property BeforeRequest :`

Visibility: published

Access:

Description: Published version of TCustomFPWebModule.BeforeRequest (1122).

See also: TCustomFPWebModule.BeforeRequest (1122)

33.8.6 TFPWebModule.OnRequest

Synopsis: Event called for request processing

Declaration: `Property OnRequest :`

Visibility: published

Access:

Description: Published version of TCustomFPWebModule.OnRequest (1123).

See also: TCustomFPWebModule.OnRequest (1123)

33.8.7 TFPWebModule.AfterResponse

Synopsis: Event called after response processing

Declaration: `Property AfterResponse :`

Visibility: published

Access:

Description: Published version of TCustomFPWebModule.AfterResponse (1123).

See also: TCustomFPWebModule.AfterResponse (1123)

33.8.8 TFPWebModule.OnGetAction

Synopsis: Event called to determine which action to execute

Declaration: `Property OnGetAction :`

Visibility: published

Access:

Description: Published version of TCustomFPWebModule.OnGetAction (1123).

See also: TCustomFPWebModule.OnGetAction (1123)

33.8.9 TFPWebModule.DefActionWhenUnknown

Synopsis: Whether to use the default action when no specific action is found

Declaration: `Property DefActionWhenUnknown :`

Visibility: published

Access:

Description: Published version of TCustomFPWebModule.DefActionWhenUnknown (1123).

See also: TCustomFPWebModule.DefActionWhenUnknown (1123)

33.8.10 TFPWebModule.CreateSession

Synopsis: Whether to automatically create sessions

Declaration: `Property CreateSession :`

Visibility: published

Access:

Description: Determines whether sessions should be automatically created for incoming requests. This property is inherited from the parent HTTP module class.

See also: Session (1131)

33.8.11 TFPWebModule.Session

Synopsis: The current session object

Declaration: `Property Session :`

Visibility: published

Access:

Description: Provides access to the current session object for the request. This property is inherited from the parent HTTP module class.

See also: CreateSession (1131)

33.8.12 TFPWebModule.Kind

Synopsis: The module kind

Declaration: `Property Kind :`

Visibility: published

Access:

Description: Specifies the kind of web module. This property is inherited from the parent HTTP module class.

33.8.13 TFPWebModule.OnNewSession

Synopsis: Event called when a new session is created

Declaration: `Property OnNewSession :`

Visibility: published

Access:

Description: Event triggered when a new session is created for a client. This property is inherited from the parent HTTP module class.

See also: OnSessionExpired (1132)

33.8.14 TFPWebModule.OnSessionExpired

Synopsis: Event called when a session expires

Declaration: `Property OnSessionExpired :`

Visibility: published

Access:

Description: Event triggered when a session expires or is explicitly terminated. This property is inherited from the parent HTTP module class.

See also: OnNewSession (1132)

33.8.15 TFPWebModule.AfterInitModule

Synopsis: Event called after module initialization

Declaration: `Property AfterInitModule :`

Visibility: published

Access:

Description: Event triggered after the module has been initialized. This property is inherited from the parent HTTP module class.

33.8.16 TFPWebModule.CORS

Synopsis: Cross-Origin Resource Sharing configuration

Declaration: `Property CORS :`

Visibility: published

Access:

Description: Configuration object for Cross-Origin Resource Sharing (CORS) support. This property is inherited from the parent HTTP module class.

33.9 TTemplateVar

33.9.1 Description

`TTemplateVar` represents a single template variable that can be used in template processing. Each variable has a name and a corresponding value that will be substituted during template processing.

See also: TTemplateVars (1134), TFPTemplate (1119)

33.9.2 Method overview

Page	Method	Description
1133	Assign	Assigns properties from another template variable
1133	GetDisplayName	Returns the display name for this template variable

33.9.3 Property overview

Page	Properties	Access	Description
1134	Name	rw	The name of the template variable
1134	Value	rw	The value of the template variable

33.9.4 TTemplateVar.Assign

Synopsis: Assigns properties from another template variable

Declaration: `procedure Assign(Source: TPersistent);  Override`

Visibility: public

Description: Copies the name and value from another `TTemplateVar` instance to this one.

33.9.5 TTemplateVar.GetDisplayName

Synopsis: Returns the display name for this template variable

Declaration: `function GetDisplayName : string;  Override`

Visibility: public

Description: Returns the variable name as the display name. This is used for displaying the variable in collection editors and similar contexts.

See also: Name (1134)

33.9.6 TTemplateVar.Name

Synopsis: The name of the template variable

Declaration: `Property Name : string`

Visibility: published

Access: Read,Write

Description: The name used to identify this variable in templates. This name is used for variable substitution during template processing.

See also: Value (1134)

33.9.7 TTemplateVar.Value

Synopsis: The value of the template variable

Declaration: `Property Value : string`

Visibility: published

Access: Read,Write

Description: The text value that will be substituted when the variable name is encountered in template processing.

See also: Name (1134)

33.10 TTemplateVars

33.10.1 Description

`TTemplateVars` manages a collection of TTemplateVar (1133) objects. It provides methods to find variables by name and access them by index.

This collection is used by web modules to maintain template variables that can be referenced during template processing.

See also: TTemplateVar (1133)

33.10.2 Method overview

Page	Method	Description
1135	FindVar	Searches for a variable with the specified name
1135	IndexOfVar	Returns the index of the variable with the specified name
1135	VarByName	Returns the variable with the specified name

33.10.3 Property overview

Page	Properties	Access	Description
1135	Variables	rw	Array property providing indexed access to template variables

33.10.4 TTemplateVars.IndexOfVar

Synopsis: Returns the index of the variable with the specified name

Declaration: `function IndexOfVar(AName: string) : Integer`

Visibility: public

Description: Searches for a template variable with the given name and returns its index in the collection. Returns -1 if no variable with the specified name is found.

See also: VarByName (1135), FindVar (1135)

33.10.5 TTemplateVars.VarByName

Synopsis: Returns the variable with the specified name

Declaration: `function VarByName(AName: string) : TTemplateVar`

Visibility: public

Description: Finds and returns the template variable with the given name. Raises an exception if no variable with the specified name is found.

Errors: Raises EFPWebError (1120) if the variable name is not found.

See also: FindVar (1135), EFPWebError (1120)

33.10.6 TTemplateVars.FindVar

Synopsis: Searches for a variable with the specified name

Declaration: `function FindVar(AName: string) : TTemplateVar`

Visibility: public

Description: Searches for a template variable with the given name. Returns `nil` if no variable with the specified name is found.

See also: VarByName (1135)

33.10.7 TTemplateVars.Variables

Synopsis: Array property providing indexed access to template variables

Declaration: `Property Variables[I: Integer]: TTemplateVar; default`

Visibility: public

Access: Read,Write

Description: Provides indexed access to the template variables in the collection. This is the default array property for the class.

See also: TTemplateVar (1133)

Chapter 34

Reference for unit 'gettext'

34.1 Used units

Table 34.1: Used units by unit 'gettext'

Name	Page
Classes	??
System	??
sysutils	??

34.2 Overview

The gettext unit can be used to hook into the resource string mechanism of Free Pascal to provide translations of the resource strings, based on the GNU gettext mechanism. The unit provides a class (TMOFile (1139)) to read the .mo files with localizations for various languages. It also provides a couple of calls to translate all resource strings in an application based on the translations in a .mo file.

34.3 Constants, types and variables

34.3.1 Constants

```
MOFileHeaderMagic = $950412DE
```

This constant is found as the first integer in a .mo

34.3.2 Types

```
PLongWordArray = ^TLongWordArray
```

Pointer to a TLongWordArray (1137) array.

```
PMOStringTable = ^TMOStringTable
```

Pointer to a TMOStringTable (1137) array.

```
PPCharArray = ^TPCharArray
```

Pointer to a TPCharArray (1137) array.

```
TLongWordArray = Array[0..(1shl30)divSizeOf(LongWord)] of
  LongWord
```

TLongWordArray is an array used to define the PLongWordArray (1136) pointer. A variable of type TLongWordArray should never be directly declared, as it would occupy too much memory. The PLongWordArray type can be used to allocate a dynamic number of elements.

```
TMOStringTable = Array[0..(1shl30)divSizeOf(TMOStringInfo)] of
  TMOStringInfo
```

TMOStringTable is an array type containing TMOStringInfo (1139) records. It should never be used directly, as it would occupy too much memory.

```
TPCharArray = Array[0..(1shl30)divSizeOf(PAnsiChar)] of PAnsiChar
```

TLongWordArray is an array used to define the PPCharArray (1137) pointer. A variable of type TPCharArray should never be directly declared, as it would occupy too much memory. The PPCharArray type can be used to allocate a dynamic number of elements.

```
TTranslationErrorHandler = procedure(const aFileName: string;
  const aUnitName: string;
  aError: Exception;
  out ReRaise: Boolean)
```

34.3.3 Variables

```
OnTranslationError : TTranslationErrorHandler = Nil
```

34.4 Procedures and functions

34.4.1 GetLanguageIDs

Synopsis: Return the current language IDs.

Declaration:
```
procedure GetLanguageIDs(var Lang: AnsiString;
                         var FallbackLang: AnsiString)
```

Visibility: default

Description: GetLanguageIDs returns the current language IDs (an ISO string) as returned by the operating system. On windows, the GetUserDefaultLCID and GetLocaleInfo calls are used. On other operating systems, the LC_ALL, LC_MESSAGES or LANG environment variables are examined.

34.4.2 TranslateResourceStrings

Synopsis: Translate the resource strings of the application.

Declaration:
```
procedure TranslateResourceStrings(AFile: TMOFile)
procedure TranslateResourceStrings(const AFilename: AnsiString)
```

Visibility: default

Description: `TranslateResourceStrings` translates all the resource strings in the application based on the values in the .mo file `AFileName` or `AFile`. The procedure creates an TMOFile (1139) instance to read the .mo file if a filename is given.

Errors: If the file does not exist or is an invalid .mo file.

See also: TranslateUnitResourceStrings (1138), TMOFile (1139)

34.4.3 TranslateUnitResourceStrings

Synopsis: Translate the resource strings of a unit.

Declaration:
```
procedure TranslateUnitResourceStrings(const AUnitName: AnsiString;
                                       AFile: TMOFile)
procedure TranslateUnitResourceStrings(const AUnitName: AnsiString;
                                       const AFilename: AnsiString)
```

Visibility: default

Description: `TranslateUnitResourceStrings` is identical in function to TranslateResourceStrings (1138), but translates the strings of a single unit (`AUnitName`) which was used to compile the application. This can be more convenient, since the resource string files are created on a unit basis.

See also: TranslateResourceStrings (1138), TMOFile (1139)

34.5 TMOFileHeader

```
TMOFileHeader = packed record
  magic : LongWord;
  revision : LongWord
  ;
  nstrings : LongWord;
  OrigTabOffset : LongWord;
  TransTabOffset
  : LongWord;
  HashTabSize : LongWord;
  HashTabOffset : LongWord
  ;
end
```

This structure describes the structure of a .mo file with string localizations.

34.6 TMOStringInfo

```
TMOStringInfo = packed record
  &length : LongWord;
  offset : LongWord
  ;
end
```

This record is one element in the string tables describing the original and translated strings. It describes the position and length of the string. The location of these tables is stored in the TMOFileHeader (1138) record at the start of the file.

34.7 EMOFileError

34.7.1 Description

`EMOFileError` is raised in case an TMOFile (1139) instance is created with an invalid .mo.

See also: TMOFile (1139)

34.8 TMOFile

34.8.1 Description

`TMOFile` is a class providing easy access to a .mo file. It can be used to translate any of the strings that reside in the .mo file. The internal structure of the .mo is completely hidden.

34.8.2 Method overview

Page	Method	Description
1139	Create	Create a new instance of the `TMOFile` class.
1140	Destroy	Removes the `TMOFile` instance from memory.
1140	Translate	Translate a string.

34.8.3 TMOFile.Create

Synopsis: Create a new instance of the `TMOFile` class.

Declaration:
```
constructor Create(const AFilename: string)
constructor Create(AStream: TStream)
```

Visibility: public

Description: `Create` creates a new instance of the `MOFile` class. It opens the file `AFileName` or the stream `AStream`. If a stream is provided, it should be seekable.

The whole contents of the file is read into memory during the `Create` call. This means that the stream is no longer needed after the `Create` call.

Errors: If the named file does not exist, then an exception may be raised. If the file does not contain a valid TMOFileHeader (1138) structure, then an EMOFileError (1139) exception is raised.

See also: TMOFile.Destroy (1140)

34.8.4 TMOFile.Destroy

Synopsis: Removes the `TMOFile` instance from memory.

Declaration:
```
destructor Destroy;  Override
```

Visibility: public

Description: `Destroy` cleans the internal structures with the contents of the .mo. After this the `TMOFile` instance is removed from memory.

See also: TMOFile.Create (1139)

34.8.5 TMOFile.Translate

Synopsis: Translate a string.

Declaration:
```
function Translate(AOrig: PAnsiChar; ALen: Integer; AHash: LongWord)
                     : AnsiString
function Translate(const AOrig: AnsiString; AHash: LongWord)
                     : AnsiString
function Translate(const AOrig: AnsiString) : AnsiString
```

Visibility: public

Description: `Translate` translates the string `AOrig`. The string should be in the .mo file as-is. The string can be given as a plain string, as a `PChar` (with length `ALen`). If the hash value (`AHash`) of the string is not given, it is calculated.

If the string is in the `.mo` file, the translated string is returned. If the string is not in the file, an empty string is returned.

Errors: None.

Chapter 35

Reference for unit 'HTTPDefs'

35.1 Used units

Table 35.1: Used units by unit 'HTTPDefs'

Name	Page
Classes	??
httpprotocol	1215
System	??
sysutils	??
TypInfo	??
URIParser	1610

35.2 Overview

The `HTTPDefs` unit provides essential types, constants, and classes for HTTP protocol handling. It contains basic HTTP header (1174) field definitions, request (1197) and response (1203) classes, cookie management (1163), file upload (1210) support, and MIME (1193) content handling for web applications.

35.3 Using TRequest and TResponse Classes

The `TRequest` and `TResponse` classes are the foundation of HTTP communication in FCL-Web. `TRequest` represents incoming HTTP requests with methods, headers, parameters, and content. `TResponse` represents outgoing HTTP responses with status codes, headers, and content.

These classes provide comprehensive access to HTTP protocol features including custom headers, content types, query parameters, form data, and file uploads.

```
program RequestResponseExample;

{$mode objfpc}{$H+}

uses
```

```
  httpdefs, SysUtils, Classes;

procedure ProcessRequest(Request: TRequest; Response: TResponse);
var
  UserAgent, ContentType: String;
  Stream: TStringStream;
begin
  Writeln('=== Processing HTTP Request ===');

  // Read request information
  Writeln('Method: ', Request.Method);
  Writeln('URI: ', Request.URI);
  Writeln('Query String: ', Request.QueryString);
  Writeln('Content Length: ', Request.ContentLength);

  // Read headers
  UserAgent := Request.GetCustomHeader('User-Agent');
  if UserAgent <> '' then
    Writeln('User Agent: ', UserAgent);

  // Read query parameters
  if Request.QueryFields.Values['search'] <> '' then
    Writeln('Search query: ', Request.QueryFields.Values['search']);

  // Read POST data (if any)
  if Request.ContentLength > 0 then
  begin
    Writeln('Request content: ', Request.Content);
    ContentType := Request.ContentType;
    Writeln('Content type: ', ContentType);
  end;

  // Process based on request method
  case Request.Method of
    'GET':
      begin
        Response.Content := '{"message": "GET request processed"}';
        Response.ContentType := 'application/json';
        Response.Code := 200;
      end;
    'POST':
      begin
        // Echo back the posted data
        Response.Content := 'Received: ' + Request.Content;
        Response.ContentType := 'text/plain';
        Response.Code := 201;
      end;
    'PUT':
      begin
        Response.Content := 'Resource updated';
        Response.Code := 200;
      end;
    'DELETE':
      begin
```

```
        Response.Content := '';
        Response.Code := 204; // No Content
      end;
  else
    Response.Content := 'Method not allowed';
    Response.Code := 405;
  end;

  // Set custom response headers
  Response.SetCustomHeader('X-Powered-By', 'FreePascal-FCL-Web');
  Response.SetCustomHeader('X-Request-ID', 'req-' + IntToStr(Random(99999)));

  // Set security headers
  Response.SetCustomHeader('X-Content-Type-Options', 'nosniff');
  Response.SetCustomHeader('X-Frame-Options', 'DENY');

  Writeln('=== Response Generated ===');
  Writeln('Status Code: ', Response.Code);
  Writeln('Content Type: ', Response.ContentType);
  Writeln('Content Length: ', Length(Response.Content));
end;

procedure DemonstrateStreamContent(Response: TResponse);
var
  Stream: TStringStream;
begin
  Writeln;
  Writeln('=== Demonstrating Stream Content ===');

  Stream := TStringStream.Create('{"users": ["alice", "bob", "charlie"]}');
  try
    Response.ContentStream := Stream;
    Response.ContentType := 'application/json';
    Response.FreeContentStream := False; // We manage the stream

    Writeln('Content set via stream');
    Writeln('Stream size: ', Stream.Size, ' bytes');
  finally
    Stream.Free;
  end;
end;

var
  Request: TRequest;
  Response: TResponse;
begin
  Request := TRequest.Create;
  Response := TResponse.Create(Request);
  try
    // Simulate a GET request
    Request.Method := 'GET';
    Request.URI := '/api/users';
    Request.QueryString := 'search=john&limit=10';
    Request.QueryFields.Values['search'] := 'john';
```

```
    Request.QueryFields.Values['limit'] := '10';
    Request.SetCustomHeader('User-Agent', 'TestClient/1.0');

    ProcessRequest(Request, Response);

    // Simulate a POST request
    Writeln;
    Request.Method := 'POST';
    Request.URI := '/api/users';
    Request.Content := '{"name": "John Doe", "email": "john@example.com"}';
    Request.ContentType := 'application/json';

    ProcessRequest(Request, Response);

    // Demonstrate stream content
    DemonstrateStreamContent(Response);

  finally
    Response.Free;
    Request.Free;
  end;
end.
```

35.4 Working with HTTP Cookies

HTTP cookies provide a way to store small pieces of data in the client's browser. The `TCookie` class represents individual cookies, while `TCookies` manages collections of cookies. Cookies can have various attributes like expiration time, domain, path, and security settings.

Cookies are automatically sent by browsers with subsequent requests to the same domain, making them useful for session management, user preferences, and tracking.

```
program CookieExample;

{$mode objfpc}{$H+}

uses
  httpdefs, SysUtils, DateUtils;

procedure DemonstrateCookies(Response: TResponse);
var
  SessionCookie, PrefCookie, SecureCookie: TCookie;
begin
  // Create a session cookie (expires when browser closes)
  SessionCookie := Response.Cookies.Add;
  SessionCookie.Name := 'session_id';
  SessionCookie.Value := 'abc123xyz789';
  SessionCookie.Path := '/';

  // Create a preference cookie (expires in 30 days)
  PrefCookie := Response.Cookies.Add;
  PrefCookie.Name := 'user_theme';
  PrefCookie.Value := 'dark_mode';
```

```
  PrefCookie.Path := '/';
  PrefCookie.Domain := 'example.com';
  PrefCookie.Expires := Now + 30; // 30 days from now

  // Create a secure cookie for sensitive data
  SecureCookie := Response.Cookies.Add;
  SecureCookie.Name := 'auth_token';
  SecureCookie.Value := 'secure_token_here';
  SecureCookie.Path := '/admin';
  SecureCookie.Secure := True;     // Only sent over HTTPS
  SecureCookie.HttpOnly := True;   // Not accessible via JavaScript
  SecureCookie.SameSite := ssStrict; // CSRF protection
  SecureCookie.Expires := Now + (1/24); // Expires in 1 hour

  Writeln('Cookies set in response:');
  Writeln('1. Session cookie: ', SessionCookie.AsString);
  Writeln('2. Preference cookie: ', PrefCookie.AsString);
  Writeln('3. Secure cookie: ', SecureCookie.AsString);
end;

procedure ReadCookiesFromRequest(Request: TRequest);
var
  SessionID, Theme: String;
begin
  // Read cookies from request (simulate browser sending cookies)
  SessionID := Request.CookieFields.Values['session_id'];
  Theme := Request.CookieFields.Values['user_theme'];

  if SessionID <> '' then
    Writeln('Found session ID: ', SessionID)
  else
    Writeln('No session cookie found');

  if Theme <> '' then
    Writeln('User theme preference: ', Theme)
  else
    Writeln('No theme preference found');
end;

var
  Request: TRequest;
  Response: TResponse;
  Cookie: TCookie;
begin
  Request := TRequest.Create;
  Response := TResponse.Create(Request);
  try
    // Demonstrate setting cookies
    DemonstrateCookies(Response);

    // Simulate reading cookies from a request
    Writeln;
    Writeln('=== Simulating cookie reading ===');
    Request.CookieFields.Values['session_id'] := 'abc123xyz789';
```

```
    Request.CookieFields.Values['user_theme'] := 'dark_mode';
    ReadCookiesFromRequest(Request);

    // Demonstrate cookie expiration
    Writeln;
    Writeln('=== Cookie management ===');
    Cookie := Response.Cookies.FindCookie('session_id');
    if Cookie <> nil then
    begin
      Writeln('Expiring session cookie...');
      Cookie.Expire; // Sets expiration to past date
      Writeln('Cookie after expiration: ', Cookie.AsString);
    end;
  finally
    Response.Free;
    Request.Free;
  end;
end.
```

35.5 Understanding TCORSSupport.HandleRequest Return Value

The `TCORSSupport.HandleRequest` method returns a boolean value that indicates whether the request should continue processing after CORS validation. Understanding this return value is crucial for proper CORS implementation.

`True` - The request passes CORS validation and should be processed normally. This means the origin is allowed, the method is permitted, and required headers are present.

`False` - The request fails CORS validation and should be rejected. The method automatically sets appropriate error response headers when returning `False`.

```
program CORSReturnValueExample;

{$mode objfpc}{$H+}

uses
  httpdefs, SysUtils;

procedure ProcessRequestWithCORS(Request: TRequest; Response: TResponse);
var
  CORS: TCORSSupport;
  AllowRequest: Boolean;
begin
  CORS := TCORSSupport.Create;
  try
    CORS.Enabled := True;
    CORS.AllowedOrigins := 'https://trusted-site.com';
    CORS.AllowedMethods := 'GET,POST';

    // Handle CORS - the return value determines next steps
    AllowRequest := CORS.HandleRequest(Request, Response, []);

    if AllowRequest then
    begin
```

```
      Writeln('[OK] CORS validation passed - processing request');

      // Continue with normal request processing
      if Request.Method = 'GET' then
      begin
        Response.Content := 'Data from server';
        Response.ContentType := 'text/plain';
        Response.Code := 200;
      end
      else if Request.Method = 'POST' then
      begin
        Response.Content := 'Data received';
        Response.Code := 201;
      end
      else if Request.Method = 'OPTIONS' then
      begin
        // Preflight handled by CORS support
        Response.Code := 200;
      end;
    end
    else
    begin
      Writeln('[KO] CORS validation failed - request blocked');

      // Don't process the request - CORS already set error response
      // Response.Code and headers are already set by HandleRequest
      Writeln('Response already configured with CORS error');
    end;
  finally
    CORS.Free;
  end;
end;

var
  Request: TRequest;
  Response: TResponse;
begin
  Request := TRequest.Create;
  Response := TResponse.Create(Request);
  try
    // Test with allowed origin
    Writeln('=== Testing with allowed origin ===');
    Request.Method := 'GET';
    Request.SetCustomHeader('Origin', 'https://trusted-site.com');
    ProcessRequestWithCORS(Request, Response);

    // Reset for next test
    Response.Code := 200;
    Response.Content := '';

    // Test with disallowed origin
    Writeln('=== Testing with disallowed origin ===');
    Request.SetCustomHeader('Origin', 'https://malicious-site.com');
    ProcessRequestWithCORS(Request, Response);
```

```
    Writeln('Final response code: ', Response.Code);
  finally
    Response.Free;
    Request.Free;
  end;
end.
```

35.6 CORS Support Usage

Cross-Origin Resource Sharing (CORS) allows web applications running at one origin to access resources from another origin. The `TCORSSupport` class handles CORS headers and validation.

CORS support includes configuration of allowed origins, methods, and headers, as well as handling preflight OPTIONS requests. The `HandleRequest` method processes CORS requests and returns `True` if the request should continue or `False` if it should be blocked.

```
program CORSExample;

{$mode objfpc}{$H+}

uses
  httpdefs, SysUtils;

procedure HandleCORSRequest(Request: TRequest; Response: TResponse);
var
  CORS: TCORSSupport;
  ContinueProcessing: Boolean;
begin
  CORS := TCORSSupport.Create;
  try
    // Configure CORS settings
    CORS.Enabled := True;
    CORS.AllowedOrigins := 'https://example.com,https://app.example.com';
    CORS.AllowedMethods := 'GET,POST,PUT,DELETE,OPTIONS';
    CORS.AllowedHeaders := 'Content-Type,Authorization,X-Requested-With';
    CORS.MaxAge := 3600; // Cache preflight response for 1 hour

    // Handle the CORS request
    ContinueProcessing := CORS.HandleRequest(Request, Response, []);

    if ContinueProcessing then
    begin
      // Process the actual request
      if Request.Method = 'GET' then
      begin
        Response.Content := '{"message": "CORS request handled successfully"}';
        Response.ContentType := 'application/json';
      end
      else if Request.Method = 'OPTIONS' then
      begin
        // Preflight request - CORS headers already set
        Response.Code := 204; // No Content
```

```
      end;

      Writeln('CORS request from origin: ', Request.GetCustomHeader('Origin'));
      Writeln('Request processed successfully');
    end
    else
    begin
      // CORS validation failed
      Response.Code := 403; // Forbidden
      Response.Content := 'CORS policy violation';
      Writeln('CORS request blocked');
    end;
  finally
    CORS.Free;
  end;
end;

var
  Request: TRequest;
  Response: TResponse;
begin
  Request := TRequest.Create;
  Response := TResponse.Create(Request);
  try
    // Simulate a CORS request
    Request.Method := 'GET';
    Request.SetCustomHeader('Origin', 'https://example.com');

    HandleCORSRequest(Request, Response);

    Writeln('Response code: ', Response.Code);
    Writeln('Response content: ', Response.Content);
  finally
    Response.Free;
    Request.Free;
  end;
end.
```

35.7 Working with HTTP Sessions

HTTP sessions provide a way to maintain state across multiple HTTP requests from the same client. The `TCustomSession` class manages session data using session IDs that are typically stored in cookies.

Sessions can be in different states: `ssNew` (newly created), `ssActive` (currently valid), `ssExpired` (timed out), or `ssResponseInitialized` (response has been started). Session variables are accessed using string key-value pairs through the `Variables` property.

```
program SessionExample;

{$mode objfpc}{$H+}

uses
```

```
  httpdefs, SysUtils, Classes;

type
  TMySession = class(TCustomSession)
  private
    FData: TStringList;
  protected
    function GetSessionVariable(const VarName: String): String; override;
    procedure SetSessionVariable(const VarName: String; const AValue: String); overr
  public
    constructor Create(AOwner: TComponent); override;
    destructor Destroy; override;
  end;

constructor TMySession.Create(AOwner: TComponent);
begin
  inherited Create(AOwner);
  FData := TStringList.Create;
  TimeOutMinutes := 30; // 30 minutes timeout
end;

destructor TMySession.Destroy;
begin
  FData.Free;
  inherited Destroy;
end;

function TMySession.GetSessionVariable(const VarName: String): String;
var
  Index: Integer;
begin
  Index := FData.IndexOfName(VarName);
  if Index >= 0 then
    Result := FData.ValueFromIndex[Index]
  else
    Result := '';
end;

procedure TMySession.SetSessionVariable(const VarName: String; const AValue: String)
begin
  FData.Values[VarName] := AValue;
end;

var
  Session: TMySession;
begin
  Session := TMySession.Create(nil);
  try
    // Store session data
    Session.Variables['username'] := 'john_doe';
    Session.Variables['last_login'] := DateTimeToStr(Now);

    // Retrieve session data
    Writeln('Username: ', Session.Variables['username']);
```

```
    Writeln('Last login: ', Session.Variables['last_login']);

    Writeln('Session ID: ', Session.SessionID);
    Writeln('Session cookie: ', Session.SessionCookie);
  finally
    Session.Free;
  end;
end.
```

35.8 Constants, types and variables

35.8.1 Constants

```
DefaultAllowedHeaders =
  'x-requested-with, content-type, authorization'
```

`DefaultAllowedHeaders` contains the default set of HTTP headers allowed in CORS requests. This constant provides a standard list of commonly permitted headers for cross-origin requests.

```
DefaultAllowedMethods = 'GET, PUT, POST, OPTIONS, HEAD'
```

`DefaultAllowedMethods` contains the default set of HTTP methods allowed in CORS requests. This constant provides a standard list of permitted methods for cross-origin resource sharing.

```
DefaultAllowedOrigins = '*'
```

`DefaultAllowedOrigins` contains the default set of origins allowed for CORS requests. This constant provides a standard list of permitted origins for cross-origin resource sharing.

```
DefaultTimeOut = 15
```

`DefaultTimeOut` specifies the default timeout value of 15 seconds for HTTP operations.

```
fieldAccept = HeaderAcceptdeprecated
```

`fieldAccept` contains the HTTP header field name for content type acceptance. This constant is deprecated in favour of using the httpprotocol unit constants.

```
FieldAcceptCharset = HeaderAcceptCharsetdeprecated
```

`FieldAcceptCharset` contains the HTTP header field name for character set acceptance. This constant is deprecated in favour of using the httpprotocol unit constants.

```
FieldAcceptEncoding = HeaderAcceptEncodingdeprecated
```

`FieldAcceptEncoding` contains the HTTP header field name for encoding acceptance. This constant is deprecated in favour of using the httpprotocol unit constants.

```
FieldAcceptLanguage = HeaderAcceptLanguagedeprecated
```

`FieldAcceptLanguage` contains the HTTP header field name for language preference specification. This constant is deprecated in favour of using the httpprotocol unit constants.

```
FieldAcceptRanges = HeaderAcceptRangesdeprecated
```

FieldAcceptRanges contains the HTTP header field name for range request acceptance. This constant is deprecated in favour of using the httpprotocol unit constants.

```
FieldAge = HeaderAgedeprecated
```

FieldAge contains the HTTP header field name for response age specification. This constant is deprecated in favour of using the httpprotocol unit constants.

```
FieldAllow = HeaderAllowdeprecated
```

FieldAllow contains the HTTP header field name for allowed methods specification. This constant is deprecated in favour of using the httpprotocol unit constants.

```
FieldAuthorization = HeaderAuthorizationdeprecated
```

FieldAuthorization contains the HTTP header field name for authorization credentials. This constant is deprecated in favour of using the httpprotocol unit constants.

```
FieldCacheControl = HeaderCacheControldeprecated
```

FieldCacheControl contains the HTTP header field name for cache control directives. This constant is deprecated in favour of using the httpprotocol unit constants.

```
FieldConnection = HeaderConnectiondeprecated
```

FieldConnection contains the HTTP header field name for connection management. This constant is deprecated in favour of using the httpprotocol unit constants.

```
FieldContentEncoding = HeaderContentEncodingdeprecated
```

FieldContentEncoding contains the HTTP header field name for content encoding specification. This constant is deprecated in favour of using the httpprotocol unit constants.

```
FieldContentLanguage = HeaderContentLanguagedeprecated
```

FieldContentLanguage contains the HTTP header field name for content language specification. This constant is deprecated in favour of using the httpprotocol unit constants.

```
FieldContentLength = HeaderContentLengthdeprecated
```

FieldContentLength contains the HTTP header field name for content length specification. This constant is deprecated in favour of using the httpprotocol unit constants.

```
FieldContentLocation = HeaderContentLocationdeprecated
```

FieldContentLocation contains the HTTP header field name for content location specification. This constant is deprecated in favour of using the httpprotocol unit constants.

```
FieldContentMD5 = HeaderContentMD5deprecated
```

FieldContentMD5 contains the HTTP header field name for content MD5 checksum. This constant is deprecated in favour of using the httpprotocol unit constants.

FieldContentRange = HeaderContentRangedeprecated

FieldContentRange contains the HTTP header field name for content range specification. This constant is deprecated in favour of using the httpprotocol unit constants.

FieldContentType = HeaderContentTypedeprecated

FieldContentType contains the HTTP header field name for content type specification. This constant is deprecated in favour of using the httpprotocol unit constants.

FieldCookie = HeaderCookiedeprecated

FieldCookie contains the HTTP header field name for cookie transmission. This constant is deprecated in favour of using the httpprotocol unit constants.

FieldDate = HeaderDatedeprecated

FieldDate contains the HTTP header field name for date specification. This constant is deprecated in favour of using the httpprotocol unit constants.

FieldETag = HeaderETagdeprecated

FieldETag contains the HTTP header field name for entity tag specification. This constant is deprecated in favour of using the httpprotocol unit constants.

FieldExpect = HeaderExpectdeprecated

FieldExpect contains the HTTP header field name for expectation specification. This constant is deprecated in favour of using the httpprotocol unit constants.

FieldExpires = HeaderExpiresdeprecated

FieldExpires contains the HTTP header field name for expiration date specification. This constant is deprecated in favour of using the httpprotocol unit constants.

FieldFrom = HeaderFromdeprecated

FieldFrom contains the HTTP header field name for sender email address specification. This constant is deprecated in favour of using the httpprotocol unit constants.

FieldHost = HeaderHostdeprecated

FieldHost contains the HTTP header field name for host header specification. This constant is deprecated in favour of using the httpprotocol unit constants.

FieldIfMatch = HeaderIfMatchdeprecated

FieldIfMatch contains the HTTP header field name for conditional request matching. This constant is deprecated in favour of using the httpprotocol unit constants.

```
FieldIfModifiedSince = HeaderIfModifiedSincedeprecated
```

FieldIfModifiedSince contains the HTTP header field name for conditional modification checking. This constant is deprecated in favour of using the httpprotocol unit constants.

```
FieldIfNoneMatch = HeaderIfNoneMatchdeprecated
```

FieldIfNoneMatch contains the HTTP header field name for conditional request non-matching. This constant is deprecated in favour of using the httpprotocol unit constants.

```
FieldIfRange = HeaderIfRangedeprecated
```

FieldIfRange contains the HTTP header field name for conditional range request. This constant is deprecated in favour of using the httpprotocol unit constants.

```
FieldIfUnModifiedSince = HeaderIfUnModifiedSincedeprecated
```

FieldIfUnModifiedSince contains the HTTP header field name for conditional unmodification checking. This constant is deprecated in favour of using the httpprotocol unit constants.

```
FieldLastModified = HeaderLastModifieddeprecated
```

FieldLastModified contains the HTTP header field name for last modification date specification. This constant is deprecated in favour of using the httpprotocol unit constants.

```
FieldLocation = HeaderLocationdeprecated
```

FieldLocation contains the HTTP header field name for resource location specification. This constant is deprecated in favour of using the httpprotocol unit constants.

```
FieldMaxForwards = HeaderMaxForwardsdeprecated
```

FieldMaxForwards contains the HTTP header field name for maximum forwards specification. This constant is deprecated in favour of using the httpprotocol unit constants.

```
FieldPragma = HeaderPragmadeprecated
```

FieldPragma contains the HTTP header field name for pragma directive specification. This constant is deprecated in favour of using the httpprotocol unit constants.

```
FieldProxyAuthenticate = HeaderProxyAuthenticatedeprecated
```

FieldProxyAuthenticate contains the HTTP header field name for proxy authentication specification. This constant is deprecated in favour of using the httpprotocol unit constants.

```
FieldProxyAuthorization = HeaderProxyAuthorizationdeprecated
```

FieldProxyAuthorization contains the HTTP header field name for proxy authorization specification. This constant is deprecated in favour of using the httpprotocol unit constants.

```
FieldRange = HeaderRangedeprecated
```

FieldRange contains the HTTP header field name for range request specification. This constant is deprecated in favour of using the httpprotocol unit constants.

```
FieldReferer = HeaderRefererdeprecated
```

FieldReferer contains the HTTP header field name for referrer URL specification. This constant is deprecated in favour of using the httpprotocol unit constants.

```
FieldRetryAfter = HeaderRetryAfterdeprecated
```

FieldRetryAfter contains the HTTP header field name for retry delay specification. This constant is deprecated in favour of using the httpprotocol unit constants.

```
FieldServer = HeaderServerdeprecated
```

FieldServer contains the HTTP header field name for server information specification. This constant is deprecated in favour of using the httpprotocol unit constants.

```
FieldSetCookie = HeaderSetCookiedeprecated
```

FieldSetCookie contains the HTTP header field name for setting cookies in responses. This constant is deprecated in favour of using the httpprotocol unit constants.

```
FieldTE = HeaderTEdeprecated
```

FieldTE contains the HTTP header field name for transfer encoding preference. This constant is deprecated in favour of using the httpprotocol unit constants.

```
FieldTrailer = HeaderTrailerdeprecated
```

FieldTrailer contains the HTTP header field name for trailer header specification. This constant is deprecated in favour of using the httpprotocol unit constants.

```
FieldTransferEncoding = HeaderTransferEncodingdeprecated
```

FieldTransferEncoding contains the HTTP header field name for transfer encoding specification. This constant is deprecated in favour of using the httpprotocol unit constants.

```
FieldUpgrade = HeaderUpgradedeprecated
```

FieldUpgrade contains the HTTP header field name for protocol upgrade specification. This constant is deprecated in favour of using the httpprotocol unit constants.

```
FieldUserAgent = HeaderUserAgentdeprecated
```

FieldUserAgent contains the HTTP header field name for user agent identification. This constant is deprecated in favour of using the httpprotocol unit constants.

```
FieldVary = HeaderVarydeprecated
```

FieldVary contains the HTTP header field name for response variation specification. This constant is deprecated in favour of using the httpprotocol unit constants.

```
FieldVia = HeaderViadeprecated
```

FieldVia contains the HTTP header field name for proxy path specification. This constant is deprecated in favour of using the httpprotocol unit constants.

```
FieldWarning = HeaderWarningdeprecated
```

FieldWarning contains the HTTP header field name for warning message specification. This constant is deprecated in favour of using the httpprotocol unit constants.

```
FieldWWWAuthenticate = HeaderWWWAuthenticatedeprecated
```

FieldWWWAuthenticate contains the HTTP header field name for authentication requirement specification. This constant is deprecated in favour of using the httpprotocol unit constants.

```
FieldXRequestedWith = HeaderXRequestedWithdeprecated
```

FieldXRequestedWith contains the HTTP header field name for request origin identification. This constant is deprecated in favour of using the httpprotocol unit constants.

```
HeaderBasedVariables = [hvCookie, hvSetCookie, hvXRequestedWith]
```

HeaderBasedVariables contains a set of HTTP variable types that are derived from HTTP headers. This constant identifies which variables correspond to standard HTTP header fields.

```
HTTPDateFmt = httpProtocol.HTTPDateFmt
```

HTTPDateFmt contains the standard HTTP date format string used for formatting dates in HTTP headers. This is an alias to the constant in the httpprotocol unit.

```
HTTPDays : Array[1..7] of string = ('Sun', 'Mon', 'Tue', 'Wed', 'Thu'
  , 'Fri', 'Sat')
```

HTTPDays is an array containing 3-letter abbreviations for days of the week (Sun, Mon, Tue, etc.) used in HTTP date formatting.

```
HTTPFieldIndexes : THttpIndexes = (1, 2, 3, 4, 5, 6, 7, 8, 9, 10,
  11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 34, 0, 36
  , 26)deprecated
```

HTTPFieldIndexes is an array that maps HTTP variable types to their corresponding field indexes. This provides efficient lookup of header field positions by variable type.

```
HTTPFieldNames : THttpFields = (fieldAccept, fieldAcceptCharset, fieldAcceptEncoding
  , fieldAcceptLanguage, fieldAuthorization, fieldConnection, fieldContentEncoding
  , fieldContentLanguage, fieldContentLength, fieldContentType, fieldCookie
  , fieldDate, fieldExpires, fieldFrom, fieldIfModifiedSince, fieldLastModified
  , fieldLocation, fieldPragma, fieldReferer, fieldRetryAfter, fieldServer
  , fieldSetCookie, fieldUserAgent, fieldWWWAuthenticate, fieldHost
  , fieldCacheControl, fieldXRequestedWith, fieldIfNoneMatch
  )deprecated
```

HTTPFieldNames is an array containing the names of standard HTTP header fields. This array provides mapping from field indexes to their string representations.

```
HTTPMonths : Array[1..12] of string = ('Jan', 'Feb', 'Mar', 'Apr'
  , 'May', 'Jun', 'Jul', 'Aug', 'Sep', 'Oct', 'Nov', 'Dec')
```

HTTPMonths is an array containing 3-letter abbreviations for months (Jan, Feb, Mar, etc.) used in HTTP date formatting.

```
NoHTTPFields = 28
```

NoHTTPFields contains the value 28, representing the number of standard HTTP header fields. This constant is deprecated.

```
SCookieDomain = httpProtocol.SCookieDomain
```

SCookieDomain contains the string 'domain' used in cookie domain attributes. This constant specifies the domain scope for cookies.

```
SCookieExpire = httpProtocol.SCookieExpire
```

SCookieExpire contains the string 'expires' used in cookie attributes. This is an alias to the constant in the httpprotocol unit.

```
SCookieHttpOnly = httpProtocol.SCookieHttpOnly
```

SCookieHttpOnly contains the string 'HttpOnly' used in cookie HttpOnly attributes. This constant indicates that cookies should not be accessible via JavaScript.

```
SCookiePath = httpProtocol.SCookiePath
```

SCookiePath contains the string 'path' used in cookie path attributes. This constant specifies the path scope for cookies.

```
SCookieSameSite = httpProtocol.SCookieSameSite
```

SCookieSameSite contains the string 'SameSite' used in cookie SameSite attributes. This constant controls cross-site request behavior for cookies.

```
SCookieSecure = httpProtocol.SCookieSecure
```

SCookieSecure contains the string 'secure' used in cookie secure attributes. This constant indicates that cookies should only be sent over secure connections.

```
SFPWebSession = 'FPWebSession'
```

SFPWebSession contains the string 'FPWebSession' which is used as the default name for session cookies in web applications.

35.8.2 Types

```
HTTPError = EHTTP
```

`HTTPError` creates an EHTTP (1162) exception with the specified status code and message. It provides a convenient way to raise HTTP-specific errors with proper status information.

```
TCORSOption = (coAllowCredentials,coEmptyDomainToOrigin)
```

Table 35.2: Enumeration values for type TCORSOption

Value	Explanation
coAllowCredentials	Allow credentials in CORS requests
coEmptyDomainToOrigin	Convert empty domain to origin in CORS

`TCORSOption` defines individual options for CORS configuration. These options control various aspects of cross-origin resource sharing behavior.

```
TCORSOptions = Set of TCORSOption
```

`TCORSOptions` is a set of TCORSOption (1158) values. It allows combining multiple CORS configuration options for comprehensive control.

```
THandleCORSOption = (hcDetect,hcFull,hcHumanReadable,hcSend)
```

Table 35.3: Enumeration values for type THandleCORSOption

Value	Explanation
hcDetect	Detect CORS requests option
hcFull	Full CORS handling option
hcHumanReadable	Human readable CORS handling option
hcSend	Send CORS headers option

`THandleCORSOption` defines options that control how CORS requests are handled. These options affect the processing and response generation for cross-origin requests.

```
THandleCORSOptions = Set of THandleCORSOption
```

`THandleCORSOptions` is a set of THandleCORSOption (1158) values. It allows configuring multiple aspects of CORS request handling behavior.

```
THttpFields = Array[1..NoHTTPFields] of String
```

`THttpFields` is an array type containing HTTP header field names. This array maps field indexes to their corresponding header field name strings.

```
THttpIndexes = Array[1..NoHTTPFields] of Integer
```

`THttpIndexes` is an array type containing HTTP header field index mappings. This array provides reverse lookup from header field names to their indexes.

```
THTTPVariables = Array[THTTPVariableType] of string
```

`THTTPVariables` is an array type that maps HTTP variable types to their corresponding values. This provides indexed access to HTTP request and environment variables.

```
THTTPVariableType = (hvUnknown,hvHTTPVersion,hvMethod,hvCookie,
  hvSetCookie,hvXRequestedWith,hvPathInfo,
  hvPathTranslated,hvRemoteAddress,hvRemoteHost,
  hvScriptName,hvServerPort,hvURL,hvQuery,hvContent
  )
```

Table 35.4: Enumeration values for type THTTPVariableType

Value	Explanation
hvContent	Request body content
hvCookie	Cookie header from request
hvHTTPVersion	HTTP protocol version
hvMethod	HTTP request method
hvPathInfo	PATH_INFO environment variable
hvPathTranslated	PATH_TRANSLATED environment variable
hvQuery	QUERY_STRING environment variable
hvRemoteAddress	REMOTE_ADDR environment variable
hvRemoteHost	REMOTE_HOST environment variable
hvScriptName	SCRIPT_NAME environment variable
hvServerPort	SERVER_PORT environment variable
hvSetCookie	Set-Cookie header field variable
hvUnknown	Unknown or unspecified variable type
hvURL	Complete request URL variable
hvXRequestedWith	X-Requested-With header field variable

`THTTPVariableType` defines the different types of HTTP-related variables that can be handled by the web server. These include request information, headers, and CGI environment variables.

hvUnknown Unknown or unspecified variable type

hvHTTPVersion HTTP protocol version

hvMethod HTTP request method (GET, POST, etc.)

hvCookie Cookie header from request

hvSetCookie Set-Cookie header for response

hvXRequestedWith X-Requested-With header (AJAX indicator)

hvPathInfo Additional path information

hvPathTranslated Translated path information

hvRemoteAddress Client IP address

hvRemoteHost Client hostname

hvScriptName Script name being executed

hvServerPort Server port number

hvURL Request URL

hvQuery Query string parameters

hvContent Request body content

```
THTTPVariableTypes = Set of THTTPVariableType
```

`THTTPVariableTypes` is a set type based on `THTTPVariableType` enumeration. It allows specification of multiple HTTP variable types simultaneously.

```
TMimeItemClass = Class of TMimeItem
```

`TMimeItemClass` is a class reference type for creating instances of `TMimeItem` or its descendants. This enables customization of MIME item processing.

```
TMimeItemsClass = Class of TMimeItems
```

`TMimeItemsClass` is a class reference type for creating instances of `TMimeItems` or its descendants. This enables customization of MIME items collection handling.

```
TOnUnknownEncodingEvent = procedure(Sender: TRequest;
  const ContentType: string;
  Stream: TStream) of object
```

`TOnUnknownEncodingEvent` is an event type triggered when unknown content encoding is encountered. It allows custom handling of content with unrecognized encoding through the provided stream.

```
TRequestEvent = procedure(Sender: TObject; ARequest: TRequest) of
  object
```

`TRequestEvent` is an event type for handling HTTP request processing. It provides access to the request object during various stages of request handling.

```
TResponseEvent = procedure(Sender: TObject; AResponse: TResponse)
  of object
```

`TResponseEvent` is an event type for handling HTTP response processing. It provides access to the response object during various stages of response generation.

```
TSameSite = (ssEmpty,ssNone,ssStrict,ssLax)
```

Table 35.5: Enumeration values for type TSameSite

Value	Explanation
ssEmpty	No SameSite attribute
ssLax	Send with top-level navigation
ssNone	Send with all cross-site requests
ssStrict	Send only with same-site requests

TSameSite defines the possible values for the SameSite cookie attribute, which controls when cookies are sent with cross-site requests.

ssEmpty No SameSite attribute specified

ssNone Cookie sent with all cross-site requests

ssStrict Cookie only sent with same-site requests

ssLax Cookie sent with top-level navigation cross-site requests

```
TSessionState = (ssNew,ssExpired,ssActive,ssResponseInitialized)
```

Table 35.6: Enumeration values for type TSessionState

Value	Explanation
ssActive	Active session state
ssExpired	Expired session state
ssNew	New session state
ssResponseInitialized	Response initialized session state

TSessionState defines the possible states of an HTTP session. This enumeration tracks the lifecycle of user sessions from creation to expiration.

```
TSessionStates = Set of TSessionState
```

TSessionStates represents a set of TSessionState (1161) values. It is used to track multiple session states simultaneously during session management.

```
TUploadedFileClass = Class of TUploadedFile
```

TUploadedFileClass is a class reference type that allows creating instances of TUploadedFile or its descendants. This enables customization of uploaded file handling by using derived classes.

```
TUploadedFilesClass = Class of TUploadedFiles
```

TUploadedFilesClass is a class reference type for creating instances of TUploadedFiles or its descendants. This enables customization of uploaded files collection handling.

35.8.3 Variables

```
MimeItemClass : TMimeItemClass = Nil
```

MimeItemClass is a global class reference variable for TMimeItem (1193). It allows customization of the MIME item class used for individual content items.

```
MimeItemsClass : TMimeItemsClass = TMimeItems
```

MimeItemsClass is a global class reference variable for TMimeItems (1196). It allows customization of the MIME items collection class used for content processing.

```
UploadedFileClass : TUploadedFileClass = TUploadedFile
```

`UploadedFileClass` is a global class reference variable for TUploadedFile (1210). It allows customization of the uploaded file class used for individual file uploads.

```
UploadedFilesClass : TUploadedFilesClass = TUploadedFiles
```

`UploadedFilesClass` is a global class reference variable for TUploadedFiles (1212). It allows customization of the uploaded files collection class used throughout the HTTP framework.

35.9 Procedures and functions

35.9.1 HTTPDecode

Synopsis: Decodes URL-encoded string

Declaration: `function HTTPDecode(const AStr: string) : string`

Visibility: default

Description: `HTTPDecode` converts a URL-encoded string back to its original form, replacing percent-encoded characters with their actual values.

See also: HTTPEncode (1162)

35.9.2 HTTPEncode

Synopsis: Encodes string for URL transmission

Declaration: `function HTTPEncode(const AStr: string) : string`

Visibility: default

Description: `HTTPEncode` converts a string to URL-encoded format, replacing special characters with percent-encoded representations suitable for HTTP transmission.

See also: HTTPDecode (1162)

35.9.3 IncludeHTTPPathDelimiter

Synopsis: Ensures path ends with forward slash

Declaration: `function IncludeHTTPPathDelimiter(const AStr: string) : string`

Visibility: default

Description: `IncludeHTTPPathDelimiter` appends a forward slash ('/') to the end of a path string if it doesn't already end with one, ensuring proper HTTP path formatting.

35.10 EHTTP

35.10.1 Description

`EHTTP` is an exception class for HTTP-related errors. It includes properties for HTTP status code and status text.

See also: HTTPError (1158)

35.10.2 Property overview

Page	Properties	Access	Description
1163	StatusCode	rw	HTTP status code of the error
1163	StatusText	rw	HTTP status text of the error

35.10.3 EHTTP.StatusCode

Synopsis: HTTP status code of the error

Declaration: `Property StatusCode : Integer`

Visibility: public

Access: Read,Write

Description: `StatusCode` contains the HTTP status code associated with the error. This numeric code indicates the type of HTTP error that occurred.

35.10.4 EHTTP.StatusText

Synopsis: HTTP status text of the error

Declaration: `Property StatusText : string`

Visibility: public

Access: Read,Write

Description: `StatusText` contains the textual description of the HTTP status code. This provides a human-readable explanation of the error condition.

35.11 TCookie

35.11.1 Description

`TCookie` represents an HTTP cookie with all its attributes. It provides properties for name, value, domain, path, expiration, security settings, and other cookie attributes.

See also: TCookies (1167), TSameSite (1160)

35.11.2 Method overview

Page	Method	Description
1164	Assign	Assigns values from another persistent object
1164	Create	Creates a new cookie instance
1164	Expire	Marks the cookie as expired

35.11.3 Property overview

Page	Properties	Access	Description
1166	AsString	r	String representation of the cookie
1165	Domain	rw	Cookie domain restriction
1165	Expires	rw	Expiration date and time of the cookie
1166	HttpOnly	rw	HTTP-only access flag
1164	Name	rw	Cookie name
1165	Path	rw	Path attribute of the cookie
1166	SameSite	rw	SameSite attribute of the cookie
1166	Secure	rw	Cookie security flag
1165	Value	rw	Cookie value

35.11.4 TCookie.Create

Synopsis: Creates a new cookie instance

Declaration: `constructor Create(ACollection: TCollection);  Override`

Visibility: public

Description: `Create` initializes a new `TCookie` instance as part of the specified collection.

See also: TCookies (1167)

35.11.5 TCookie.Assign

Synopsis: Assigns values from another persistent object

Declaration: `procedure Assign(Source: TPersistent);  Override`

Visibility: public

Description: `Assign` copies all cookie properties from the source `TCookie` object to this instance.

35.11.6 TCookie.Expire

Synopsis: Marks the cookie as expired

Declaration: `procedure Expire`

Visibility: public

Description: `Expire` sets the cookie's expiration date to a past date, causing the client to delete the cookie.

See also: Expires (1165)

35.11.7 TCookie.Name

Synopsis: Cookie name

Declaration: `Property Name : string`

Visibility: public

Access: Read,Write

Description: `Name` specifies the name of the cookie. This is the identifier used to reference the cookie.

See also: Value (1165)

35.11.8 TCookie.Value

Synopsis: Cookie value

Declaration: `Property Value : string`

Visibility: public

Access: Read,Write

Description: `Value` contains the data stored in the cookie. This is the information that will be sent back by the client.

See also: Name (1164)

35.11.9 TCookie.Domain

Synopsis: Cookie domain restriction

Declaration: `Property Domain : string`

Visibility: public

Access: Read,Write

Description: `Domain` specifies which domain the cookie applies to. If empty, the cookie applies only to the current domain.

See also: Path (1165)

35.11.10 TCookie.Path

Synopsis: Path attribute of the cookie

Declaration: `Property Path : string`

Visibility: public

Access: Read,Write

Description: `Path` specifies the URL path that must exist in the requested URL for the browser to send the cookie. If not specified, defaults to the path of the request URL that generated the cookie.

35.11.11 TCookie.Expires

Synopsis: Expiration date and time of the cookie

Declaration: `Property Expires : TDateTime`

Visibility: public

Access: Read,Write

Description: `Expires` specifies when the cookie should expire. After this date and time, the browser will delete the cookie automatically.

35.11.12 TCookie.Secure

Synopsis: Cookie security flag

Declaration: `Property Secure : Boolean`

Visibility: public

Access: Read,Write

Description: `Secure` determines whether the cookie should only be sent over secure (HTTPS) connections. When `True`, the cookie will not be sent over unencrypted HTTP connections.

See also: HttpOnly (1166), SameSite (1166)

35.11.13 TCookie.HttpOnly

Synopsis: HTTP-only access flag

Declaration: `Property HttpOnly : Boolean`

Visibility: public

Access: Read,Write

Description: `HttpOnly` when `True` prevents client-side scripts from accessing the cookie, reducing the risk of cross-site scripting (XSS) attacks.

See also: Secure (1166)

35.11.14 TCookie.SameSite

Synopsis: SameSite attribute of the cookie

Declaration: `Property SameSite : TSameSite`

Visibility: public

Access: Read,Write

Description: `SameSite` controls whether cookies are sent with cross-site requests. This attribute helps prevent cross-site request forgery attacks.

35.11.15 TCookie.AsString

Synopsis: String representation of the cookie

Declaration: `Property AsString : string`

Visibility: public

Access: Read

Description: `AsString` returns the complete cookie as a string formatted according to HTTP cookie specifications. This includes the name, value, and all attributes formatted for transmission in HTTP headers.

35.12 TCookies

35.12.1 Description

`TCookies` is a collection class that manages multiple `TCookie` objects. It provides methods to add, find, and access cookies by name or index.

See also: TCookie (1163)

35.12.2 Method overview

Page	Method	Description
1167	Add	Adds a new cookie to the collection
1167	CookieByName	Finds a cookie by name
1167	FindCookie	Finds a cookie by name
1168	IndexOfCookie	Returns the index of a cookie by name

35.12.3 Property overview

Page	Properties	Access	Description
1168	Items	rw	Cookie at the specified index

35.12.4 TCookies.Add

Synopsis: Adds a new cookie to the collection

Declaration: `function Add : TCookie`

Visibility: public

Description: `Add` creates and adds a new `TCookie` instance to the collection, returning a reference to it.

See also: AddFromString (1167)

35.12.5 TCookies.CookieByName

Synopsis: Finds a cookie by name

Declaration: `function CookieByName(AName: string) : TCookie`

Visibility: public

Description: `CookieByName` searches for and returns a cookie with the specified name. If no cookie is found, it returns `nil`.

See also: FindCookie (1167), IndexOfCookie (1168)

35.12.6 TCookies.FindCookie

Synopsis: Finds a cookie by name

Declaration: `function FindCookie(AName: string) : TCookie`

Visibility: public

Description: `FindCookie` searches for a cookie with the specified name and returns it. Returns `nil` if no cookie with the given name is found.

35.12.7 TCookies.IndexOfCookie

Synopsis: Returns the index of a cookie by name

Declaration: `function IndexOfCookie(AName: string) : Integer`

Visibility: public

Description: `IndexOfCookie` returns the zero-based index of the cookie with the specified name. Returns -1 if no cookie with the given name is found.

35.12.8 TCookies.Items

Synopsis: Cookie at the specified index

Declaration: `Property Items[Index: Integer]: TCookie; default`

Visibility: public

Access: Read,Write

Description: `Items` provides indexed access to cookies in the collection. This is the default array property for the `TCookies` class.

35.13 TCORSSupport

35.13.1 Description

`TCORSSupport` provides functionality for handling CORS (Cross-Origin Resource Sharing) requests. It manages allowed origins, methods, headers, and other CORS-related settings.

See also: TCORSOptions (1158), THandleCORSOptions (1158)

35.13.2 Method overview

Page	Method	Description
1169	Assign	Assigns CORS settings from another object
1169	Create	Creates a new CORS support instance
1169	HandleRequest	Handles CORS request processing
1169	ResolvedCORSAllowedOrigins	Resolves allowed origins for CORS requests

35.13.3 Property overview

Page	Properties	Access	Description
1170	AllowedHeaders	rw	Headers allowed for CORS requests
1170	AllowedMethods	rw	HTTP methods allowed for CORS requests
1170	AllowedOrigins	rw	Origins allowed for CORS requests
1169	Enabled	rw	Enables or disables CORS support
1171	MaxAge	rw	Maximum age for CORS preflight cache in seconds
1170	Options	rw	CORS configuration options

35.13.4 TCORSSupport.Create

Synopsis: Creates a new CORS support instance

Declaration:
```
constructor Create;  Virtual
```

Visibility: public

Description: `Create` constructs a new CORS support instance with the specified owner. It initializes the CORS configuration with default values.

35.13.5 TCORSSupport.ResolvedCORSAllowedOrigins

Synopsis: Resolves allowed origins for CORS requests

Declaration:
```
function ResolvedCORSAllowedOrigins(aRequest: TRequest) : string
                                          ;  Virtual
```

Visibility: public

Description: `ResolvedCORSAllowedOrigins` resolves the list of allowed origins for the current request. It processes the configured origins and returns the applicable ones based on the request context.

35.13.6 TCORSSupport.HandleRequest

Synopsis: Handles CORS request processing

Declaration:
```
function HandleRequest(aRequest: TRequest; aResponse: TResponse;
                           aOptions: THandleCORSOptions) : Boolean;  Virtual
```

Visibility: public

Description: `HandleRequest` processes CORS headers and validation for the HTTP request. It returns the appropriate CORS handling result based on the request and configuration.

35.13.7 TCORSSupport.Assign

Synopsis: Assigns CORS settings from another object

Declaration:
```
procedure Assign(Source: TPersistent);  Override
```

Visibility: public

Description: `Assign` copies CORS configuration settings from another `TCORSSupport` instance. It transfers all CORS options, origins, methods, and headers to the current instance.

35.13.8 TCORSSupport.Enabled

Synopsis: Enables or disables CORS support

Declaration:
```
Property Enabled : Boolean
```

Visibility: published

Access: Read,Write

Description: `Enabled` controls whether CORS processing is active. When `True`, CORS headers are processed and validated for cross-origin requests.

35.13.9 TCORSSupport.Options

Synopsis: CORS configuration options

Declaration: `Property Options : TCORSOptions`

Visibility: published

Access: Read,Write

Description: `Options` contains the set of CORS configuration options (1158). These options control various aspects of CORS behavior such as credential handling.

35.13.10 TCORSSupport.AllowedMethods

Synopsis: HTTP methods allowed for CORS requests

Declaration: `Property AllowedMethods : string`

Visibility: published

Access: Read,Write

Description: `AllowedMethods` specifies the HTTP methods permitted for cross-origin requests. Common methods include GET, POST, PUT, DELETE, and OPTIONS.

35.13.11 TCORSSupport.AllowedOrigins

Synopsis: Origins allowed for CORS requests

Declaration: `Property AllowedOrigins : string`

Visibility: published

Access: Read,Write

Description: `AllowedOrigins` contains the list of origins permitted to make cross-origin requests. Origins can be specific domains or wildcard patterns for broader access.

35.13.12 TCORSSupport.AllowedHeaders

Synopsis: Headers allowed for CORS requests

Declaration: `Property AllowedHeaders : string`

Visibility: published

Access: Read,Write

Description: `AllowedHeaders` specifies the HTTP headers that are permitted in cross-origin requests. This controls which headers clients can include in their CORS requests.

35.13.13 TCORSSupport.MaxAge

Synopsis: Maximum age for CORS preflight cache in seconds

Declaration: `Property MaxAge : Integer`

Visibility: published

Access: Read,Write

Description: `MaxAge` specifies how long browsers may cache CORS preflight responses. This value is sent in the Access-Control-Max-Age header to control preflight caching.

35.14 TCustomSession

35.14.1 Description

`TCustomSession` provides the foundation for HTTP session management functionality. It handles session initialization, variable management, state tracking, and cleanup operations.

35.14.2 Method overview

Page	Method	Description
1171	Create	Creates a new session instance
1172	InitResponse	Initializes the HTTP response for the session
1172	InitSession	Initializes the session from HTTP request
1172	RemoveVariable	Removes a variable from the session
1173	SessionVariableExists	Checks if a session variable exists
1172	Terminate	Terminates and invalidates the session
1172	UpdateResponse	Updates the HTTP response with session information

35.14.3 Property overview

Page	Properties	Access	Description
1174	OnSessionStateChange	rw	Event triggered when session state changes
1173	SessionCookie	rw	Name of the cookie used to store session ID
1174	SessionCookiePath	rw	Path attribute for the session cookie
1173	SessionID	r	Unique identifier for the session
1174	SessionState	r	Current state of the session
1173	TimeOutMinutes	rw	Session timeout period in minutes
1174	Variables	rw	Access to session variables by name

35.14.4 TCustomSession.Create

Synopsis: Creates a new session instance

Declaration: `constructor Create(AOwner: TComponent);  Override`

Visibility: public

Description: `Create` constructs a new session instance with the specified owner. It initializes the session with default values and prepares it for use.

35.14.5 TCustomSession.InitSession

Synopsis: Initializes the session from HTTP request

Declaration:
```
procedure InitSession(ARequest: TRequest; OnNewSession: TNotifyEvent;
                      OnExpired: TNotifyEvent);  Virtual
```

Visibility: public

Description: `InitSession` initializes the session using data from the HTTP request. It reads session cookies and triggers the new session event handler if needed.

35.14.6 TCustomSession.InitResponse

Synopsis: Initializes the HTTP response for the session

Declaration:
```
procedure InitResponse(AResponse: TResponse);  Virtual
```

Visibility: public

Description: `InitResponse` initializes the HTTP response for session handling. It sets up response headers and cookies required for session management.

35.14.7 TCustomSession.UpdateResponse

Synopsis: Updates the HTTP response with session information

Declaration:
```
procedure UpdateResponse(AResponse: TResponse);  Virtual;  Abstract
```

Visibility: public

Description: `UpdateResponse` updates the HTTP response with current session information. It ensures session cookies and headers are properly set in the response.

35.14.8 TCustomSession.RemoveVariable

Synopsis: Removes a variable from the session

Declaration:
```
procedure RemoveVariable(VariableName: string);  Virtual;  Abstract
```

Visibility: public

Description: `RemoveVariable` removes the specified variable from the session storage. The variable name is case-insensitive for removal operations.

35.14.9 TCustomSession.Terminate

Synopsis: Terminates and invalidates the session

Declaration:
```
procedure Terminate;  Virtual;  Abstract
```

Visibility: public

Description: `Terminate` terminates the current session and cleans up session data. It marks the session for deletion and removes associated cookies from the response.

35.14.10 TCustomSession.SessionVariableExists

Synopsis: Checks if a session variable exists

Declaration:
```
function SessionVariableExists(VarName: string) : Boolean;  Virtual
                                ;  Abstract
```

Visibility: public

Description: `SessionVariableExists` checks whether a variable with the specified name exists in the session. Returns `True` if the variable exists, `False` otherwise.

35.14.11 TCustomSession.TimeOutMinutes

Synopsis: Session timeout period in minutes

Declaration:
```
Property TimeOutMinutes : Integer
```

Visibility: public

Access: Read,Write

Description: `TimeOutMinutes` specifies the session timeout period in minutes. Sessions that remain inactive longer than this period will expire automatically.

35.14.12 TCustomSession.SessionID

Synopsis: Unique identifier for the session

Declaration:
```
Property SessionID : string
```

Visibility: public

Access: Read

Description: `SessionID` contains the unique identifier for the current session. This identifier is used to track and maintain session state across HTTP requests.

35.14.13 TCustomSession.SessionCookie

Synopsis: Name of the cookie used to store session ID

Declaration:
```
Property SessionCookie : string
```

Visibility: public

Access: Read,Write

Description: `SessionCookie` specifies the name of the HTTP cookie used to store the session ID. This cookie name is used for both reading and writing session information.

35.14.14 TCustomSession.SessionCookiePath

Synopsis: Path attribute for the session cookie

Declaration: `Property SessionCookiePath : string`

Visibility: public

Access: Read,Write

Description: `SessionCookiePath` defines the path scope for the session cookie. The cookie will only be sent for requests matching this path prefix.

35.14.15 TCustomSession.Variables

Synopsis: Access to session variables by name

Declaration: `Property Variables[VarName: string]: string`

Visibility: public

Access: Read,Write

Description: `Variables` provides access to the session variables collection. It allows storing and retrieving named values that persist across HTTP requests within the session.

35.14.16 TCustomSession.SessionState

Synopsis: Current state of the session

Declaration: `Property SessionState : TSessionStates`

Visibility: public

Access: Read

Description: `SessionState` indicates the current state of the session. It tracks whether the session is new, active, or has been terminated.

35.14.17 TCustomSession.OnSessionStateChange

Synopsis: Event triggered when session state changes

Declaration: `Property OnSessionStateChange : TNotifyEvent`

Visibility: public

Access: Read,Write

Description: `OnSessionStateChange` is triggered when the session state changes. It allows monitoring and responding to session state transitions.

35.15 THTTPHeader

35.15.1 Description

`THTTPHeader` provides common functionality for handling HTTP headers, variables, and content. It serves as the base class for both `TRequest` and `TResponse` classes.

See also: TRequest (1197), TResponse (1203)

35.15.2 Method overview

Page	Method	Description
1178	Create	Creates an HTTP header instance
1178	Destroy	Destroys the HTTP header instance
1179	GetCustomHeader	Gets a custom header field value by name
1179	GetFieldByName	Gets a header field value by field name
1178	GetHeader	Gets the value of a specific header field
1179	GetVariableHeaderName	Gets the header field name for an HTTP variable type
1178	HeaderIsSet	Checks if a specific header field is set
1180	LoadFromStream	Loads HTTP headers from a stream
1180	LoadFromStrings	Loads HTTP headers from a string list
1179	SetCustomHeader	Sets a custom header field value by name
1179	SetFieldByName	Sets a header field value by field name
1178	SetHeader	Sets the value of a specific header field

35.15.3 Property overview

Page	Properties	Access	Description
1181	Accept	rw	Accept header field value
1181	AcceptCharset	rw	Accept-Charset header field value
1181	AcceptEncoding	rw	Accept-Encoding header field value
1182	AcceptLanguage	rw	Accept-Language header field value
1182	Authorization	rw	Authorization header field value
1182	Connection	rw	Connection header field value
1192	Content	rw	Request content as string
1182	ContentEncoding	rw	Content-Encoding header field value
1193	ContentFields	r	Collection of content form fields
1183	ContentLanguage	rw	Content-Language header field value
1183	ContentLength	rw	Content-Length header field value
1183	ContentType	rw	Content-Type header field value
1189	Cookie	rw	Cookie header field value
1193	CookieFields	rw	Collection of cookie name-value pairs
1193	CustomHeaders	r	Collection of custom header fields
1183	Date	rw	Date header field value
1184	Expires	rw	Expires header field value
1180	FieldCount	r	Number of header fields
1180	FieldNames	r	Name of the header field at the specified index
1180	Fields	r	Header field name and value at the specified index
1181	FieldValues	r	Value of the header field at the specified index
1184	From	rw	From header field value
1184	Host	rw	Host header field value
1187	HTTPAccept	rw	HTTP Accept header accessor property
1187	HTTPAcceptCharset	rw	HTTP Accept-Charset header accessor property
1188	HTTPAcceptEncoding	rw	HTTP Accept-Encoding header accessor property
1188	HTTPIfModifiedSince	rw	HTTP If-Modified-Since header accessor property
1188	HTTPReferer	rw	HTTP Referer header accessor property
1188	HTTPUserAgent	rw	HTTP User-Agent header accessor property
1189	HttpVersion	rw	HTTP protocol version
1189	HTTPXRequestedWith	rw	HTTP X-Requested-With header accessor property
1184	IfModifiedSince	rw	If-Modified-Since header field value
1185	LastModified	rw	Last-Modified header field value
1185	Location	rw	Location header field value
1192	Method	rw	HTTP request method
1190	PathInfo	rw	PATH_INFO environment variable value
1190	PathTranslated	rw	PATH_TRANSLATED environment variable value
1185	Pragma	rw	Pragma header field value
1190	ProtocolVersion	rw	Protocol version string
1192	Query	rw	QUERY_STRING environment variable value
1193	QueryFields	r	Collection of query string name-value pairs
1185	Referer	rw	Referer header field value
1191	RemoteAddr	rw	Alias for RemoteAddress property
1190	RemoteAddress	rw	REMOTE_ADDR environment variable value
1191	RemoteHost	rw	REMOTE_HOST environment variable value
1186	RetryAfter	rw	Retry-After header field value
1191	ScriptName	rw	SCRIPT_NAME environment variable value
1186	Server	rw	Server header field value
1191	ServerPort	rw	SERVER_PORT environment variable value
1189	SetCookie	rw	Set-Cookie header field value
1192	URL	rw	Complete request URL
1186	UserAgent	rw	User-Agent header field value
1187	Via	rw	Via header field value
1186	Warning	rw	Warning header field value
1187	WWWAuthenticate	rw	WWW-Authenticate header field value

35.15.4 THTTPHeader.Create

Synopsis: Creates an HTTP header instance

Declaration: `constructor Create; Virtual`

Visibility: public

Description: `Create` initializes a new HTTP header instance. This sets up the internal structures for managing HTTP header fields and variables.

35.15.5 THTTPHeader.Destroy

Synopsis: Destroys the HTTP header instance

Declaration: `destructor Destroy; Override`

Visibility: public

Description: `Destroy` cleans up resources used by the HTTP header instance. This includes freeing internal collections and associated objects.

35.15.6 THTTPHeader.HeaderIsSet

Synopsis: Checks if a specific header field is set

Declaration: `function HeaderIsSet(AHeader: THeader) : Boolean`

Visibility: public

Description: `HeaderIsSet` determines whether a specific HTTP header field has been set. Returns `True` if the header field contains a value, `False` otherwise.

35.15.7 THTTPHeader.GetHeader

Synopsis: Gets the value of a specific header field

Declaration: `function GetHeader(AHeader: THeader) : string`

Visibility: public

Description: `GetHeader` retrieves the value of a specific HTTP header field. Returns an empty string if the header field is not set.

35.15.8 THTTPHeader.SetHeader

Synopsis: Sets the value of a specific header field

Declaration: `procedure SetHeader(AHeader: THeader; const AValue: string)`

Visibility: public

Description: `SetHeader` assigns a value to a specific HTTP header field. This method updates the header field with the provided value.

35.15.9 THTTPHeader.SetFieldByName

Synopsis: Sets a header field value by field name

Declaration:
```
procedure SetFieldByName(const AName: string; const AValue: string)
```

Visibility: public

Description: `SetFieldByName` sets a header field value using the field name as a string. This provides a flexible way to set header fields by name rather than by enumerated type.

35.15.10 THTTPHeader.GetFieldByName

Synopsis: Gets a header field value by field name

Declaration:
```
function GetFieldByName(const AName: string) : string
```

Visibility: public

Description: `GetFieldByName` retrieves a header field value using the field name as a string. Returns an empty string if the field is not found.

35.15.11 THTTPHeader.GetVariableHeaderName

Synopsis: Gets the header field name for an HTTP variable type

Declaration:
```
class function GetVariableHeaderName(AVariable: THTTPVariableType)
                                     : string
```

Visibility: public

Description: `GetVariableHeaderName` returns the HTTP header field name corresponding to a given HTTP variable type. This provides the mapping from enumerated types to header field strings.

35.15.12 THTTPHeader.GetCustomHeader

Synopsis: Gets a custom header field value by name

Declaration:
```
function GetCustomHeader(const Name: string) : string;  Virtual
```

Visibility: public

Description: `GetCustomHeader` retrieves the value of a custom (non-standard) HTTP header field. Custom headers are typically application-specific extensions to the standard HTTP headers.

35.15.13 THTTPHeader.SetCustomHeader

Synopsis: Sets a custom header field value by name

Declaration:
```
procedure SetCustomHeader(const Name: string; const Value: string)
                          ;  Virtual
```

Visibility: public

Description: `SetCustomHeader` assigns a value to a custom (non-standard) HTTP header field. This allows setting application-specific header extensions beyond standard HTTP headers.

35.15.14 THTTPHeader.LoadFromStream

Synopsis: Loads HTTP headers from a stream

Declaration:
```
function LoadFromStream(Stream: TStream; IncludeCommand: Boolean)
                        : Integer
```

Visibility: public

Description: `LoadFromStream` parses HTTP header data from a stream and populates the header fields. This method can optionally include parsing of the command line (request/status line).

35.15.15 THTTPHeader.LoadFromStrings

Synopsis: Loads HTTP headers from a string list

Declaration:
```
function LoadFromStrings(Strings: TStrings; IncludeCommand: Boolean)
                         : Integer;  Virtual
```

Visibility: public

Description: `LoadFromStrings` parses HTTP header data from a string list and populates the header fields. Each string in the list represents a header line in standard HTTP format.

35.15.16 THTTPHeader.FieldCount

Synopsis: Number of header fields

Declaration:
```
Property FieldCount : Integer; deprecated;
```

Visibility: public

Access: Read

Description: `FieldCount` returns the total number of header fields that have been set. This includes both standard and custom header fields.

35.15.17 THTTPHeader.Fields

Synopsis: Header field name and value at the specified index

Declaration:
```
Property Fields[AIndex: Integer]: string; deprecated;
```

Visibility: public

Access: Read

Description: `Fields` provides indexed access to header field name-value pairs. Returns the complete field in the format 'Name: Value'.

35.15.18 THTTPHeader.FieldNames

Synopsis: Name of the header field at the specified index

Declaration:
```
Property FieldNames[AIndex: Integer]: string; deprecated;
```

Visibility: public

Access: Read

Description: `FieldNames` provides indexed access to header field names only. This returns just the name portion without the value.

35.15.19 THTTPHeader.FieldValues

Synopsis: Value of the header field at the specified index

Declaration: `Property FieldValues[AIndex: Integer]: string; deprecated;`

Visibility: public

Access: Read

Description: `FieldValues` provides indexed access to header field values only. This returns just the value portion without the field name.

35.15.20 THTTPHeader.Accept

Synopsis: Accept header field value

Declaration: `Property Accept : string`

Visibility: public

Access: Read,Write

Description: `Accept` contains the `Accept` header field value, which specifies the media types that are acceptable for the response. This indicates what content types the client can process.

35.15.21 THTTPHeader.AcceptCharset

Synopsis: Accept-Charset header field value

Declaration: `Property AcceptCharset : string`

Visibility: public

Access: Read,Write

Description: `AcceptCharset` contains the `Accept-Charset` header field value, which specifies the character encodings that are acceptable for the response.

35.15.22 THTTPHeader.AcceptEncoding

Synopsis: Accept-Encoding header field value

Declaration: `Property AcceptEncoding : string`

Visibility: public

Access: Read,Write

Description: `AcceptEncoding` contains the `Accept-Encoding` header field value, which specifies the content encodings that are acceptable for the response. This typically includes compression methods like gzip or deflate.

35.15.23 THTTPHeader.AcceptLanguage

Synopsis: Accept-Language header field value

Declaration: `Property AcceptLanguage : string`

Visibility: public

Access: Read,Write

Description: `AcceptLanguage` contains the `Accept-Language` header field value, which specifies the natural languages that are preferred for the response.

35.15.24 THTTPHeader.Authorization

Synopsis: Authorization header field value

Declaration: `Property Authorization : string`

Visibility: public

Access: Read,Write

Description: `Authorization` contains the `Authorization` header field value, which provides authentication credentials for the request. This typically includes the authentication scheme and credentials.

35.15.25 THTTPHeader.Connection

Synopsis: Connection header field value

Declaration: `Property Connection : string`

Visibility: public

Access: Read,Write

Description: `Connection` contains the `Connection` header field value, which specifies connection options for the current connection. Common values include 'keep-alive' and 'close'.

35.15.26 THTTPHeader.ContentEncoding

Synopsis: Content-Encoding header field value

Declaration: `Property ContentEncoding : string`

Visibility: public

Access: Read,Write

Description: `ContentEncoding` contains the `Content-Encoding` header field value, which specifies any encoding applied to the entity body. This typically indicates compression methods like gzip or deflate.

35.15.27 THTTPHeader.ContentLanguage

Synopsis: Content-Language header field value

Declaration: `Property ContentLanguage : string`

Visibility: public

Access: Read,Write

Description: `ContentLanguage` contains the `Content-Language` header field value, which specifies the natural language of the content. This indicates what language the content is written in.

35.15.28 THTTPHeader.ContentLength

Synopsis: Content-Length header field value

Declaration: `Property ContentLength : Integer`

Visibility: public

Access: Read,Write

Description: `ContentLength` contains the `Content-Length` header field value, which indicates the size of the entity body in bytes. This specifies the exact length of the message body.

35.15.29 THTTPHeader.ContentType

Synopsis: Content-Type header field value

Declaration: `Property ContentType : string`

Visibility: public

Access: Read,Write

Description: `ContentType` contains the `Content-Type` header field value, which specifies the media type of the entity body. This indicates the format and nature of the content (e.g., 'text/html', 'application/json').

35.15.30 THTTPHeader.Date

Synopsis: Date header field value

Declaration: `Property Date : string`

Visibility: public

Access: Read,Write

Description: `Date` contains the `Date` header field value, which represents the date and time when the message was sent. This is formatted according to RFC 7231 date format.

35.15.31 THTTPHeader.Expires

Synopsis: Expires header field value

Declaration: `Property Expires : string`

Visibility: public

Access: Read,Write

Description: `Expires` contains the `Expires` header field value, which specifies when the content expires and should be considered stale. This is used for caching control.

35.15.32 THTTPHeader.From

Synopsis: From header field value

Declaration: `Property From : string`

Visibility: public

Access: Read,Write

Description: `From` contains the `From` header field value, which specifies the email address of the user making the request. This is rarely used in practice due to privacy concerns.

35.15.33 THTTPHeader.Host

Synopsis: Host header field value

Declaration: `Property Host : string`

Visibility: public

Access: Read,Write

Description: `Host` contains the `Host` header field value, which specifies the hostname and port of the target server. This is required in HTTP/1.1 requests.

35.15.34 THTTPHeader.IfModifiedSince

Synopsis: If-Modified-Since header field value

Declaration: `Property IfModifiedSince : string`

Visibility: public

Access: Read,Write

Description: `IfModifiedSince` contains the `If-Modified-Since` header field value, which allows conditional requests. The server will only return the resource if it has been modified since the specified date.

35.15.35 THTTPHeader.LastModified

Synopsis: Last-Modified header field value

Declaration: `Property LastModified : string`

Visibility: public

Access: Read,Write

Description: `LastModified` contains the `Last-Modified` header field value, which indicates when the resource was last modified. This is used for caching and conditional request processing.

35.15.36 THTTPHeader.Location

Synopsis: Location header field value

Declaration: `Property Location : string`

Visibility: public

Access: Read,Write

Description: `Location` contains the `Location` header field value, which specifies a URI for redirection or location of a newly created resource. This is commonly used in redirect responses.

35.15.37 THTTPHeader.Pragma

Synopsis: Pragma header field value

Declaration: `Property Pragma : string`

Visibility: public

Access: Read,Write

Description: `Pragma` contains the `Pragma` header field value, which provides implementation-specific directives. This is primarily used for backwards compatibility with HTTP/1.0 caches.

35.15.38 THTTPHeader.Referer

Synopsis: Referer header field value

Declaration: `Property Referer : string`

Visibility: public

Access: Read,Write

Description: `Referer` contains the `Referer` header field value, which identifies the URI of the resource from which the request was made. This indicates the previous page that linked to the current request.

35.15.39 THTTPHeader.RetryAfter

Synopsis: Retry-After header field value

Declaration: `Property RetryAfter : string`

Visibility: public

Access: Read,Write

Description: `RetryAfter` contains the `Retry-After` header field value, which indicates when the client should retry the request. This is used in responses with 503 (Service Unavailable) or 429 (Too Many Requests) status codes.

35.15.40 THTTPHeader.Server

Synopsis: Server header field value

Declaration: `Property Server : string`

Visibility: public

Access: Read,Write

Description: `Server` contains the `Server` header field value, which identifies the server software handling the request. This typically includes the server name and version information.

35.15.41 THTTPHeader.UserAgent

Synopsis: User-Agent header field value

Declaration: `Property UserAgent : string`

Visibility: public

Access: Read,Write

Description: `UserAgent` contains the `User-Agent` header field value, which identifies the client application making the request. This typically includes browser name, version, and operating system information.

35.15.42 THTTPHeader.Warning

Synopsis: Warning header field value

Declaration: `Property Warning : string`

Visibility: public

Access: Read,Write

Description: `Warning` contains the `Warning` header field value, which provides additional information about the status of a message. This is used to convey warnings that might not be reflected in the status code.

35.15.43 THTTPHeader.WWWAuthenticate

Synopsis: WWW-Authenticate header field value

Declaration: `Property WWWAuthenticate : string`

Visibility: public

Access: Read,Write

Description: `WWWAuthenticate` contains the `WWW-Authenticate` header field value, which specifies the authentication schemes and parameters for the requested resource. This is sent in 401 (Unauthorized) responses to challenge the client for authentication.

35.15.44 THTTPHeader.Via

Synopsis: Via header field value

Declaration: `Property Via : string`

Visibility: public

Access: Read,Write

Description: `Via` contains the `Via` header field value, which tracks intermediate proxies and gateways. This header is used by proxies to indicate their presence in the request/response chain.

35.15.45 THTTPHeader.HTTPAccept

Synopsis: HTTP Accept header accessor property

Declaration: `Property HTTPAccept : string`

Visibility: public

Access: Read,Write

Description: `HTTPAccept` is an alias for the `Accept` property. This provides alternative access to the Accept header field value.

35.15.46 THTTPHeader.HTTPAcceptCharset

Synopsis: HTTP Accept-Charset header accessor property

Declaration: `Property HTTPAcceptCharset : string`

Visibility: public

Access: Read,Write

Description: `HTTPAcceptCharset` is an alias for the `AcceptCharset` property. This provides alternative access to the Accept-Charset header field value.

35.15.47 THTTPHeader.HTTPAcceptEncoding

Synopsis: HTTP Accept-Encoding header accessor property

Declaration: `Property HTTPAcceptEncoding : string`

Visibility: public

Access: Read,Write

Description: `HTTPAcceptEncoding` is an alias for the `AcceptEncoding` property. This provides alternative access to the Accept-Encoding header field value.

35.15.48 THTTPHeader.HTTPIfModifiedSince

Synopsis: HTTP If-Modified-Since header accessor property

Declaration: `Property HTTPIfModifiedSince : string`

Visibility: public

Access: Read,Write

Description: `HTTPIfModifiedSince` is an alias for the `IfModifiedSince` property. This provides alternative access to the If-Modified-Since header field value.

35.15.49 THTTPHeader.HTTPReferer

Synopsis: HTTP Referer header accessor property

Declaration: `Property HTTPReferer : string`

Visibility: public

Access: Read,Write

Description: `HTTPReferer` is an alias for the `Referer` property. This provides alternative access to the Referer header field value.

35.15.50 THTTPHeader.HTTPUserAgent

Synopsis: HTTP User-Agent header accessor property

Declaration: `Property HTTPUserAgent : string`

Visibility: public

Access: Read,Write

Description: `HTTPUserAgent` is an alias for the `UserAgent` property. This provides alternative access to the User-Agent header field value.

35.15.51 THTTPHeader.Cookie

Synopsis: Cookie header field value

Declaration: `Property Cookie : string`

Visibility: public

Access: Read,Write

Description: `Cookie` contains the `Cookie` header field value sent by the client. This includes all cookies that the client is sending to the server for this request.

35.15.52 THTTPHeader.SetCookie

Synopsis: Set-Cookie header field value

Declaration: `Property SetCookie : string`

Visibility: public

Access: Read,Write

Description: `SetCookie` contains the `Set-Cookie` header field value to be sent to the client. This is used by servers to send cookies to the client for storage and future requests.

35.15.53 THTTPHeader.HTTPXRequestedWith

Synopsis: HTTP X-Requested-With header accessor property

Declaration: `Property HTTPXRequestedWith : string`

Visibility: public

Access: Read,Write

Description: `HTTPXRequestedWith` contains the `X-Requested-With` header field value. This is commonly used by JavaScript frameworks to identify AJAX requests (typically set to 'XMLHttpRequest').

35.15.54 THTTPHeader.HttpVersion

Synopsis: HTTP protocol version

Declaration: `Property HttpVersion : string`

Visibility: public

Access: Read,Write

Description: `HttpVersion` contains the HTTP protocol version used in the request or response. This indicates the HTTP version such as '1.0', '1.1', or '2.0'.

35.15.55 THTTPHeader.ProtocolVersion

Synopsis: Protocol version string

Declaration: `Property ProtocolVersion : string`

Visibility: public

Access: Read,Write

Description: `ProtocolVersion` is an alias for `HttpVersion`. This provides alternative access to the HTTP protocol version information.

35.15.56 THTTPHeader.PathInfo

Synopsis: PATH_INFO environment variable value

Declaration: `Property PathInfo : string`

Visibility: public

Access: Read,Write

Description: `PathInfo` contains the PATH_INFO environment variable value. This represents additional path information following the script name in the URL.

35.15.57 THTTPHeader.PathTranslated

Synopsis: PATH_TRANSLATED environment variable value

Declaration: `Property PathTranslated : string`

Visibility: public

Access: Read,Write

Description: `PathTranslated` contains the PATH_TRANSLATED environment variable value. This represents the translated path information mapped to the file system.

35.15.58 THTTPHeader.RemoteAddress

Synopsis: REMOTE_ADDR environment variable value

Declaration: `Property RemoteAddress : string`

Visibility: public

Access: Read,Write

Description: `RemoteAddress` contains the IP address of the client making the request. This corresponds to the REMOTE_ADDR environment variable in CGI.

35.15.59 THTTPHeader.RemoteAddr

Synopsis: Alias for RemoteAddress property

Declaration: `Property RemoteAddr : string`

Visibility: public

Access: Read,Write

Description: `RemoteAddr` is an alias for the `RemoteAddress` property. This provides alternative access to the client's IP address information.

35.15.60 THTTPHeader.RemoteHost

Synopsis: REMOTE_HOST environment variable value

Declaration: `Property RemoteHost : string`

Visibility: public

Access: Read,Write

Description: `RemoteHost` contains the hostname of the client making the request. This corresponds to the REMOTE_HOST environment variable in CGI.

35.15.61 THTTPHeader.ScriptName

Synopsis: SCRIPT_NAME environment variable value

Declaration: `Property ScriptName : string`

Visibility: public

Access: Read,Write

Description: `ScriptName` contains the name of the script being executed. This corresponds to the SCRIPT_NAME environment variable in CGI.

35.15.62 THTTPHeader.ServerPort

Synopsis: SERVER_PORT environment variable value

Declaration: `Property ServerPort : Word`

Visibility: public

Access: Read,Write

Description: `ServerPort` contains the port number on which the server received the request. This corresponds to the SERVER_PORT environment variable in CGI.

35.15.63 THTTPHeader.Method

Synopsis: HTTP request method

Declaration: `Property Method : string`

Visibility: public

Access: Read,Write

Description: `Method` contains the HTTP request method used in the request. Common values include GET, POST, PUT, DELETE, HEAD, and OPTIONS.

35.15.64 THTTPHeader.URL

Synopsis: Complete request URL

Declaration: `Property URL : string`

Visibility: public

Access: Read,Write

Description: `URL` contains the complete URL of the current request. This includes the protocol, host, port, path, and query string components.

35.15.65 THTTPHeader.Query

Synopsis: QUERY_STRING environment variable value

Declaration: `Property Query : string`

Visibility: public

Access: Read,Write

Description: `Query` contains the query string portion of the request URL. This includes all parameters passed in the URL after the '?' character.

35.15.66 THTTPHeader.Content

Synopsis: Request content as string

Declaration: `Property Content : string`

Visibility: public

Access: Read,Write

Description: `Content` contains the request body content as a string. This is typically used for POST requests with text-based content types.

35.15.67 THTTPHeader.CookieFields

Synopsis: Collection of cookie name-value pairs

Declaration: `Property CookieFields : TStrings`

Visibility: public

Access: Read,Write

Description: `CookieFields` provides access to the parsed cookie values as name-value pairs. This collection contains all cookies sent by the client in the request.

35.15.68 THTTPHeader.ContentFields

Synopsis: Collection of content form fields

Declaration: `Property ContentFields : TStrings`

Visibility: public

Access: Read

Description: `ContentFields` provides access to form field values from the request content. This collection contains parsed form data from POST requests.

35.15.69 THTTPHeader.QueryFields

Synopsis: Collection of query string name-value pairs

Declaration: `Property QueryFields : TStrings`

Visibility: public

Access: Read

Description: `QueryFields` provides access to parsed query string parameters as name-value pairs. This collection contains all parameters passed in the URL query string.

35.15.70 THTTPHeader.CustomHeaders

Synopsis: Collection of custom header fields

Declaration: `Property CustomHeaders : TStringList`

Visibility: public

Access: Read

Description: `CustomHeaders` provides access to custom (non-standard) HTTP header fields. This collection contains application-specific headers beyond the standard HTTP headers.

35.16 TMimeItem

35.16.1 Description

`TMimeItem` represents a single part within a MIME multipart message. It encapsulates the content, headers, and metadata for one item in a multipart HTTP request.

35.16.2 Method overview

Page	Method	Description
1194	Process	Processes MIME data from a stream

35.16.3 Property overview

Page	Properties	Access	Description
1195	ContentType	rw	Content type of the MIME item
1194	Data	rw	Processed MIME item data
1196	DataSize	r	Size of the MIME item data in bytes
1195	Description	rw	Description of the MIME item
1195	Disposition	rw	Content disposition of the MIME item
1195	FileName	rw	Filename attribute of the MIME item
1196	IsFile	r	Indicates whether the MIME item is a file
1194	Name	rw	Name attribute of the MIME item

35.16.4 TMimeItem.Process

Synopsis: Processes MIME data from a stream

Declaration: `procedure Process(Stream: TStream); Virtual; Abstract`

Visibility: public

Description: `Process` reads and parses MIME data from the provided stream. This method processes the entire MIME item content in one operation.

35.16.5 TMimeItem.Data

Synopsis: Processed MIME item data

Declaration: `Property Data : string`

Visibility: public

Access: Read,Write

Description: `Data` contains the processed content of the MIME item. This is the decoded data after applying any content transfer encoding.

35.16.6 TMimeItem.Name

Synopsis: Name attribute of the MIME item

Declaration: `Property Name : string`

Visibility: public

Access: Read,Write

Description: `Name` contains the name attribute from the Content-Disposition header. This typically corresponds to the form field name for form uploads.

35.16.7 TMimeItem.Disposition

Synopsis: Content disposition of the MIME item

Declaration: `Property Disposition : string`

Visibility: public

Access: Read,Write

Description: `Disposition` contains the complete Content-Disposition header value. This specifies how the content should be handled (e.g., 'form-data', 'attachment').

35.16.8 TMimeItem.FileName

Synopsis: Filename attribute of the MIME item

Declaration: `Property FileName : string`

Visibility: public

Access: Read,Write

Description: `FileName` contains the filename attribute from the Content-Disposition header. This represents the original filename of an uploaded file.

35.16.9 TMimeItem.ContentType

Synopsis: Content type of the MIME item

Declaration: `Property ContentType : string`

Visibility: public

Access: Read,Write

Description: `ContentType` contains the MIME type of the item content. This indicates the format and nature of the data (e.g., 'text/plain', 'image/jpeg').

35.16.10 TMimeItem.Description

Synopsis: Description of the MIME item

Declaration: `Property Description : string`

Visibility: public

Access: Read,Write

Description: `Description` contains the Content-Description header value if present. This provides additional descriptive information about the MIME item content.

35.16.11 TMimeItem.IsFile

Synopsis: Indicates whether the MIME item is a file

Declaration: `Property IsFile : Boolean`

Visibility: public

Access: Read

Description: `IsFile` returns `True` if the MIME item represents a file upload. This is determined by the presence of a filename attribute in the Content-Disposition header.

35.16.12 TMimeItem.DataSize

Synopsis: Size of the MIME item data in bytes

Declaration: `Property DataSize : Int64`

Visibility: public

Access: Read

Description: `DataSize` indicates the total size of the MIME item data in bytes. This represents the actual size of the content after any decoding.

35.17 TMimeItems

35.17.1 Description

`TMimeItems` manages a collection of `TMimeItem` instances representing the parts of a multipart MIME message. It provides access to the individual items and handles multipart parsing.

35.17.2 Method overview

Page	Method	Description
1196	First	Returns the first MIME item in the collection
1197	Last	Returns the last MIME item in the collection

35.17.3 Property overview

Page	Properties	Access	Description
1197	Parts	r	MIME item at the specified index

35.17.4 TMimeItems.First

Synopsis: Returns the first MIME item in the collection

Declaration: `function First : TMimeItem`

Visibility: public

Description: `First` returns the first `TMimeItem` in the collection. Returns `nil` if the collection is empty.

35.17.5 TMimeItems.Last

Synopsis: Returns the last MIME item in the collection

Declaration: `function Last : TMimeItem`

Visibility: public

Description: `Last` returns the last `TMimeItem` in the collection. Returns `nil` if the collection is empty.

35.17.6 TMimeItems.Parts

Synopsis: MIME item at the specified index

Declaration: `Property Parts[AIndex: Integer]: TMimeItem; default`

Visibility: public

Access: Read

Description: `Parts` provides indexed access to `TMimeItem` instances in the collection. This is the default array property for the `TMimeItems` class.

35.18 TRequest

35.18.1 Description

`TRequest` represents an HTTP request received by a web server. It extends THTTPHeader (1174) and provides access to request data including headers, query parameters, form data, uploaded files, and other request information.

See also: TResponse (1203), THTTPHeader (1174), TUploadedFiles (1212)

35.18.2 Method overview

Page	Method	Description
1198	Create	Creates a new request instance
1198	destroy	Destroys the HTTP request instance
1198	GetNextPathInfo	Gets the next path info segment

35.18.3 Property overview

Page	Properties	Access	Description
1200	Command	r	HTTP command (method and URI)
1199	CommandLine	r	HTTP command line from request
1202	ContentRange	rw	Content-Range header field value
1201	Files	r	Collection of uploaded files
1201	HandleGetOnPost	rw	Whether to handle GET parameters on POST requests
1200	HeaderLine	r	Complete header line from request
1201	IfMatch	rw	If-Match header field value
1202	IfNoneMatch	rw	If-None-Match header field value
1202	IfRange	rw	If-Range header field value
1202	IfUnModifiedSince	rw	If-Unmodified-Since header field value
1199	LocalPathPrefix	r	Local path prefix for the request
1201	OnUnknownEncoding	rw	Event handler for unknown content encoding
1200	QueryString	rw	Query string portion of the request
1199	ReturnedPathInfo	rw	Path info that was returned from GetNextPathInfo
1199	RouteParams	rw	Route parameter value by name
1203	TE	rw	TE (Transfer-Encoding) header field value
1203	Upgrade	rw	Upgrade header field value
1200	URI	rw	URI of the HTTP request

35.18.4 TRequest.Create

Synopsis: Creates a new request instance

Declaration: `constructor Create;  Override`

Visibility: public

Description: `Create` initializes a new `TRequest` object and allocates a unique request ID.

See also: RequestID (1197)

35.18.5 TRequest.destroy

Synopsis: Destroys the HTTP request instance

Declaration: `destructor destroy;  Override`

Visibility: public

Description: `destroy` cleans up resources used by the HTTP request instance. This includes releasing uploaded files, form data, and other associated objects.

35.18.6 TRequest.GetNextPathInfo

Synopsis: Gets the next path info segment

Declaration: `function GetNextPathInfo : string`

Visibility: public

Description: `GetNextPathInfo` retrieves the next segment from the path info. This is useful for processing REST-style URLs with multiple path segments.

35.18.7 TRequest.RouteParams

Synopsis: Route parameter value by name

Declaration: `Property RouteParams[AParam: string]: string`

Visibility: public

Access: Read,Write

Description: `RouteParams` provides access to route parameters extracted from the URL path. These are typically used in REST-style routing systems to capture URL segments as parameters.

35.18.8 TRequest.ReturnedPathInfo

Synopsis: Path info that was returned from GetNextPathInfo

Declaration: `Property ReturnedPathInfo : string`

Visibility: public

Access: Read,Write

Description: `ReturnedPathInfo` contains the path info segment that was most recently returned by `GetNextPathInfo`. This allows tracking of the current position when processing path segments.

35.18.9 TRequest.LocalPathPrefix

Synopsis: Local path prefix for the request

Declaration: `Property LocalPathPrefix : string`

Visibility: public

Access: Read

Description: `LocalPathPrefix` contains the local path prefix that should be prepended to relative paths. This is useful for handling requests in subdirectories or virtual directories.

35.18.10 TRequest.CommandLine

Synopsis: HTTP command line from request

Declaration: `Property CommandLine : string`

Visibility: public

Access: Read

Description: `CommandLine` contains the complete HTTP command line from the request. This includes the method, URI, and HTTP version (e.g., 'GET /path HTTP/1.1').

35.18.11 TRequest.Command

Synopsis: HTTP command (method and URI)

Declaration: `Property Command : string`

Visibility: public

Access: Read

Description: `Command` contains the HTTP method and URI portion of the command line. This excludes the HTTP version information.

35.18.12 TRequest.URI

Synopsis: URI of the HTTP request

Declaration: `Property URI : string`

Visibility: public

Access: Read,Write

Description: `URI` contains the URI portion of the HTTP request. This includes the path and query string but excludes the protocol and host information.

35.18.13 TRequest.QueryString

Synopsis: Query string portion of the request

Declaration: `Property QueryString : string`

Visibility: public

Access: Read,Write

Description: `QueryString` contains the query string portion of the request URL. This includes all parameters passed after the '?' character in the URL.

35.18.14 TRequest.HeaderLine

Synopsis: Complete header line from request

Declaration: `Property HeaderLine : string`

Visibility: public

Access: Read

Description: `HeaderLine` contains the complete first line of the HTTP request. This is the request line including method, URI, and HTTP version.

35.18.15 TRequest.Files

Synopsis: Collection of uploaded files

Declaration: `Property Files : TUploadedFiles`

Visibility: public

Access: Read

Description: `Files` provides access to the collection of files uploaded with the request. This contains all files submitted through multipart form uploads.

35.18.16 TRequest.HandleGetOnPost

Synopsis: Whether to handle GET parameters on POST requests

Declaration: `Property HandleGetOnPost : Boolean`

Visibility: public

Access: Read,Write

Description: `HandleGetOnPost` determines whether GET parameters in the query string should be processed on POST requests. When `True`, both POST form data and GET query parameters are available.

35.18.17 TRequest.OnUnknownEncoding

Synopsis: Event handler for unknown content encoding

Declaration: `Property OnUnknownEncoding : TOnUnknownEncodingEvent`

Visibility: public

Access: Read,Write

Description: `OnUnknownEncoding` is an event that fires when an unknown content encoding is encountered. This allows custom handling of non-standard content encodings.

35.18.18 TRequest.IfMatch

Synopsis: If-Match header field value

Declaration: `Property IfMatch : string`

Visibility: public

Access: Read,Write

Description: `IfMatch` contains the `If-Match` header field value for conditional requests. This specifies entity tags that must match for the request to proceed.

35.18.19 TRequest.IfNoneMatch

Synopsis: If-None-Match header field value

Declaration: `Property IfNoneMatch : string`

Visibility: public

Access: Read,Write

Description: `IfNoneMatch` contains the `If-None-Match` header field value for conditional requests. This specifies entity tags that must not match for the request to proceed.

35.18.20 TRequest.IfRange

Synopsis: If-Range header field value

Declaration: `Property IfRange : string`

Visibility: public

Access: Read,Write

Description: `IfRange` contains the `If-Range` header field value for conditional range requests. This allows range requests only if the entity matches the specified condition.

35.18.21 TRequest.IfUnModifiedSince

Synopsis: If-Unmodified-Since header field value

Declaration: `Property IfUnModifiedSince : string`

Visibility: public

Access: Read,Write

Description: `IfUnModifiedSince` contains the `If-Unmodified-Since` header field value for conditional requests. This specifies that the request should only proceed if the resource has not been modified since the given date.

35.18.22 TRequest.ContentRange

Synopsis: Content-Range header field value

Declaration: `Property ContentRange : string`

Visibility: public

Access: Read,Write

Description: `ContentRange` contains the `Content-Range` header field value. This indicates which part of a resource is being sent in partial content responses.

35.18.23 TRequest.TE

Synopsis: TE (Transfer-Encoding) header field value

Declaration: `Property TE : string`

Visibility: public

Access: Read,Write

Description: `TE` contains the `TE` header field value, which specifies the transfer encodings the client is willing to accept. This is used in HTTP/1.1 for specifying acceptable transfer encodings.

35.18.24 TRequest.Upgrade

Synopsis: Upgrade header field value

Declaration: `Property Upgrade : string`

Visibility: public

Access: Read,Write

Description: `Upgrade` contains the `Upgrade` header field value, which is used to upgrade the connection to a different protocol. Commonly used for WebSocket upgrades.

35.19 TResponse

35.19.1 Description

`TResponse` represents an HTTP response sent back to a client. It extends `THTTPHeader` and provides methods to set status codes, headers, content, cookies, and send the response data.

See also: TRequest (1197), THTTPHeader (1174), TCookies (1167)

35.19.2 Method overview

Page	Method	Description
1204	Create	Creates an HTTP response instance
1204	destroy	Destroys the HTTP response instance
1204	SendContent	Sends the response content to the client
1205	SendHeaders	Sends the response headers to the client
1205	SendRedirect	Sends a redirect response
1205	SendResponse	Sends the complete HTTP response

35.19.3 Property overview

Page	Properties	Access	Description
1206	Age	rw	Age header value indicating response age in seconds
1206	Allow	rw	Allow header value listing permitted HTTP methods
1206	CacheControl	rw	Cache-Control header value for caching directives
1205	Code	rw	HTTP response status code
1206	CodeText	rw	HTTP response status text
1209	Content	rw	Response body content as string
1207	ContentLocation	rw	Content-Location header value for alternate location
1207	ContentMD5	rw	Content-MD5 header value with MD5 digest of content
1207	ContentRange	rw	Content-Range header value for partial content
1209	Contents	rw	String list containing response content lines
1209	ContentSent	r	Indicates whether response content has been sent
1208	ContentStream	rw	Stream containing response body content
1210	Cookies	r	Collection of HTTP cookies to send with response
1207	ETag	rw	ETag header value for entity tag validation
1208	FirstHeaderLine	rw	First line of HTTP response headers
1210	FreeContentStream	rw	Whether to automatically free the content stream
1209	HeadersSent	r	Indicates whether response headers have been sent
1208	ProxyAuthenticate	rw	Proxy-Authenticate header value for proxy authentication
1205	Request	r	Associated HTTP request object
1208	RetryAfter	rw	Retry-After header value indicating when to retry request

35.19.4 TResponse.Create

Synopsis: Creates an HTTP response instance

Declaration: `constructor Create(ARequest: TRequest); Overload`

Visibility: public

Description: `Create` initializes a new HTTP response instance associated with the specified request. This sets up the response structure and links it to the corresponding request.

35.19.5 TResponse.destroy

Synopsis: Destroys the HTTP response instance

Declaration: `destructor destroy; Override`

Visibility: public

Description: `destroy` cleans up resources used by the HTTP response instance. This includes freeing any associated content streams and releasing memory.

35.19.6 TResponse.SendContent

Synopsis: Sends the response content to the client

Declaration: `procedure SendContent`

Visibility: public

Description: `SendContent` transmits the response body content to the client. This method sends the actual data payload of the HTTP response.

35.19.7 TResponse.SendHeaders

Synopsis: Sends the response headers to the client

Declaration: `procedure SendHeaders`

Visibility: public

Description: `SendHeaders` transmits the HTTP headers to the client. This must be called before sending any content and includes the status line and all response headers.

35.19.8 TResponse.SendResponse

Synopsis: Sends the complete HTTP response

Declaration: `procedure SendResponse`

Visibility: public

Description: `SendResponse` sends both headers and content to the client, completing the HTTP response.

See also: SendHeaders (1205), SendContent (1204)

35.19.9 TResponse.SendRedirect

Synopsis: Sends a redirect response

Declaration: `procedure SendRedirect(const TargetURL: string)`

Visibility: public

Description: `SendRedirect` sends an HTTP redirect response to the client. This sets the appropriate status code and Location header to redirect the client to a new URL.

35.19.10 TResponse.Request

Synopsis: Associated HTTP request object

Declaration: `Property Request : TRequest`

Visibility: public

Access: Read

Description: `Request` contains a reference to the HTTP request object that this response is answering. This provides access to the original request data for response generation.

35.19.11 TResponse.Code

Synopsis: HTTP response status code

Declaration: `Property Code : Integer`

Visibility: public

Access: Read,Write

Description: `Code` contains the HTTP status code for the response. Common values include 200 (OK), 404 (Not Found), and 500 (Internal Server Error).

35.19.12 TResponse.CodeText

Synopsis: HTTP response status text

Declaration: `Property CodeText : string`

Visibility: public

Access: Read,Write

Description: `CodeText` contains the reason phrase corresponding to the HTTP status code. This provides the descriptive text that accompanies the numeric status code.

35.19.13 TResponse.Age

Synopsis: Age header value indicating response age in seconds

Declaration: `Property Age : string`

Visibility: public

Access: Read,Write

Description: `Age` contains the Age header value, indicating how long the response has been cached. This value is measured in seconds since the response was generated.

35.19.14 TResponse.Allow

Synopsis: Allow header value listing permitted HTTP methods

Declaration: `Property Allow : string`

Visibility: public

Access: Read,Write

Description: `Allow` contains the Allow header value, which lists the HTTP methods supported by the resource. This is commonly used in response to OPTIONS requests or with 405 Method Not Allowed responses.

35.19.15 TResponse.CacheControl

Synopsis: Cache-Control header value for caching directives

Declaration: `Property CacheControl : string`

Visibility: public

Access: Read,Write

Description: `CacheControl` contains the Cache-Control header value with caching directives. This controls how the response should be cached by browsers and intermediate caches.

35.19.16 TResponse.ContentLocation

Synopsis: Content-Location header value for alternate location

Declaration: `Property ContentLocation : string`

Visibility: public

Access: Read,Write

Description: `ContentLocation` contains the Content-Location header value. This indicates an alternate location for the content that is being returned.

35.19.17 TResponse.ContentMD5

Synopsis: Content-MD5 header value with MD5 digest of content

Declaration: `Property ContentMD5 : string`

Visibility: public

Access: Read,Write

Description: `ContentMD5` contains the Content-MD5 header value with an MD5 digest of the response content. This provides integrity checking for the transmitted content.

35.19.18 TResponse.ContentRange

Synopsis: Content-Range header value for partial content

Declaration: `Property ContentRange : string`

Visibility: public

Access: Read,Write

Description: `ContentRange` contains the Content-Range header value for partial content responses. This is used in 206 Partial Content responses to indicate which portion of the resource is being sent.

35.19.19 TResponse.ETag

Synopsis: ETag header value for entity tag validation

Declaration: `Property ETag : string`

Visibility: public

Access: Read,Write

Description: `ETag` contains the ETag header value, which is an entity tag for cache validation. This provides a mechanism for efficient cache validation and conditional requests.

35.19.20 TResponse.ProxyAuthenticate

Synopsis: Proxy-Authenticate header value for proxy authentication

Declaration: `Property ProxyAuthenticate : string`

Visibility: public

Access: Read,Write

Description: `ProxyAuthenticate` contains the Proxy-Authenticate header value. This is sent in 407 Proxy Authentication Required responses to challenge the client for proxy authentication.

35.19.21 TResponse.RetryAfter

Synopsis: Retry-After header value indicating when to retry request

Declaration: `Property RetryAfter : string`

Visibility: public

Access: Read,Write

Description: `RetryAfter` contains the Retry-After header value, indicating when the client should retry the request. This is used with 503 Service Unavailable and 429 Too Many Requests responses.

35.19.22 TResponse.FirstHeaderLine

Synopsis: First line of HTTP response headers

Declaration: `Property FirstHeaderLine : string`

Visibility: public

Access: Read,Write

Description: `FirstHeaderLine` contains the status line of the HTTP response. This includes the HTTP version, status code, and reason phrase.

35.19.23 TResponse.ContentStream

Synopsis: Stream containing response body content

Declaration: `Property ContentStream : TStream`

Visibility: public

Access: Read,Write

Description: `ContentStream` contains the response body content as a stream. This allows efficient handling of large responses without loading everything into memory.

35.19.24 TResponse.Content

Synopsis: Response body content as string

Declaration: `Property Content : RawByteString`

Visibility: public

Access: Read,Write

Description: `Content` contains the response body content as a string. This provides convenient access to the response content for text-based responses.

35.19.25 TResponse.Contents

Synopsis: String list containing response content lines

Declaration: `Property Contents : TStrings`

Visibility: public

Access: Read,Write

Description: `Contents` contains the response body content as a string list. This provides line-based access to the response content, with each line as a separate string.

35.19.26 TResponse.HeadersSent

Synopsis: Indicates whether response headers have been sent

Declaration: `Property HeadersSent : Boolean`

Visibility: public

Access: Read

Description: `HeadersSent` indicates whether the HTTP response headers have already been transmitted to the client. Once `True`, headers can no longer be modified.

35.19.27 TResponse.ContentSent

Synopsis: Indicates whether response content has been sent

Declaration: `Property ContentSent : Boolean`

Visibility: public

Access: Read

Description: `ContentSent` indicates whether the response body content has been transmitted to the client. This helps track the completion status of the response.

35.19.28 TResponse.Cookies

Synopsis: Collection of HTTP cookies to send with response

Declaration: `Property Cookies : TCookies`

Visibility: public

Access: Read

Description: `Cookies` provides access to the collection of cookies that will be sent to the client. These cookies are included in Set-Cookie headers in the HTTP response.

35.19.29 TResponse.FreeContentStream

Synopsis: Whether to automatically free the content stream

Declaration: `Property FreeContentStream : Boolean`

Visibility: public

Access: Read,Write

Description: `FreeContentStream` determines whether the content stream should be automatically freed after use. When `True`, the stream is destroyed when the response is completed.

35.20 TUploadedFile

35.20.1 Description

`TUploadedFile` encapsulates information about a file uploaded through an HTML form using multipart/form-data encoding. It provides access to the file content, metadata, and properties such as filename, size, and content type.

35.20.2 Method overview

Page	Method	Description
1211	Destroy	Destroys the uploaded file instance

35.20.3 Property overview

Page	Properties	Access	Description
1212	ContentType	rw	MIME content type of the uploaded file
1212	Description	rw	Description of the uploaded file
1212	Disposition	rw	Content disposition header of the uploaded file
1211	FieldName	rw	Form field name of the uploaded file
1211	FileName	rw	Original filename of the uploaded file
1212	LocalFileName	rw	Local temporary filename where file is stored
1211	Size	rw	Size of the uploaded file in bytes
1211	Stream	r	Stream containing the uploaded file data

35.20.4 TUploadedFile.Destroy

Synopsis: Destroys the uploaded file instance

Declaration: `destructor Destroy;  Override`

Visibility: public

Description: `Destroy` cleans up resources used by the uploaded file instance. This includes closing any associated streams and removing temporary files.

35.20.5 TUploadedFile.FieldName

Synopsis: Form field name of the uploaded file

Declaration: `Property FieldName : string`

Visibility: public

Access: Read,Write

Description: `FieldName` contains the name of the HTML form field that was used to upload the file. This corresponds to the 'name' attribute of the input element in the form.

35.20.6 TUploadedFile.FileName

Synopsis: Original filename of the uploaded file

Declaration: `Property FileName : string`

Visibility: public

Access: Read,Write

Description: `FileName` contains the original filename as specified by the client. This is the name of the file on the client's system before it was uploaded.

35.20.7 TUploadedFile.Stream

Synopsis: Stream containing the uploaded file data

Declaration: `Property Stream : TStream`

Visibility: public

Access: Read

Description: `Stream` provides access to the uploaded file content. The stream can be read to retrieve the file data or written to a local file.

35.20.8 TUploadedFile.Size

Synopsis: Size of the uploaded file in bytes

Declaration: `Property Size : Int64`

Visibility: public

Access: Read,Write

Description: `Size` indicates the total size of the uploaded file content in bytes. This represents the actual size of the data that was uploaded.

35.20.9 TUploadedFile.ContentType

Synopsis: MIME content type of the uploaded file

Declaration: `Property ContentType : string`

Visibility: public

Access: Read,Write

Description: `ContentType` contains the MIME type of the uploaded file as specified by the client. This indicates the type of content in the file (e.g., 'image/jpeg', 'text/plain').

35.20.10 TUploadedFile.Disposition

Synopsis: Content disposition header of the uploaded file

Declaration: `Property Disposition : string`

Visibility: public

Access: Read,Write

Description: `Disposition` contains the `Content-Disposition` header value from the multipart data. This typically includes information about how the content should be displayed or handled.

35.20.11 TUploadedFile.LocalFileName

Synopsis: Local temporary filename where file is stored

Declaration: `Property LocalFileName : string`

Visibility: public

Access: Read,Write

Description: `LocalFileName` contains the path to the temporary file where the uploaded content is stored on the server. This file is created during the upload process and should be moved or copied to a permanent location if needed.

35.20.12 TUploadedFile.Description

Synopsis: Description of the uploaded file

Declaration: `Property Description : string`

Visibility: public

Access: Read,Write

Description: `Description` contains additional descriptive information about the uploaded file. This may include user-provided descriptions or other metadata associated with the upload.

35.21 TUploadedFiles

35.21.1 Description

`TUploadedFiles` manages a collection of `TUploadedFile` instances. It provides methods to access, search, and manage uploaded files from HTTP requests.

35.21.2 Method overview

Page	Method	Description
1213	FileByName	Returns an uploaded file by field name
1214	FindFile	Finds an uploaded file by field name
1213	First	Returns the first uploaded file in the collection
1213	IndexOfFile	Returns the index of an uploaded file by field name
1213	Last	Returns the last uploaded file in the collection

35.21.3 Property overview

Page	Properties	Access	Description
1214	Files	rw	Uploaded file at the specified index

35.21.4 TUploadedFiles.First

Synopsis: Returns the first uploaded file in the collection

Declaration: `function First : TUploadedFile`

Visibility: public

Description: `First` returns the first `TUploadedFile` in the collection. Returns `nil` if the collection is empty.

35.21.5 TUploadedFiles.Last

Synopsis: Returns the last uploaded file in the collection

Declaration: `function Last : TUploadedFile`

Visibility: public

Description: `Last` returns the last `TUploadedFile` in the collection. Returns `nil` if the collection is empty.

35.21.6 TUploadedFiles.IndexOfFile

Synopsis: Returns the index of an uploaded file by field name

Declaration: `function IndexOfFile(AName: string) : Integer`

Visibility: public

Description: `IndexOfFile` searches for an uploaded file with the specified field name and returns its index. Returns -1 if no file with the given field name is found.

35.21.7 TUploadedFiles.FileByName

Synopsis: Returns an uploaded file by field name

Declaration: `function FileByName(AName: string) : TUploadedFile`

Visibility: public

Description: `FileByName` searches for and returns an uploaded file with the specified field name. Returns `nil` if no file with the given field name is found.

35.21.8 TUploadedFiles.FindFile

Synopsis: Finds an uploaded file by field name

Declaration: `function FindFile(AName: string) : TUploadedFile`

Visibility: public

Description: `FindFile` locates and returns an uploaded file with the specified field name. This method is an alias for `FileByName`.

35.21.9 TUploadedFiles.Files

Synopsis: Uploaded file at the specified index

Declaration: `Property Files[Index: Integer]: TUploadedFile; default`

Visibility: public

Access: Read,Write

Description: `Files` provides indexed access to uploaded files in the collection. This is the default array property for the `TUploadedFiles` class.

Chapter 36

Reference for unit 'httpprotocol'

36.1 Used units

Table 36.1: Used units by unit 'httpprotocol'

Name	Page
Classes	**??**
System	**??**
sysutils	**??**

36.2 Overview

This unit provides HTTP 1.1 protocol definitions, constants, and utility functions for web applications. It includes:

- HTTP header type definitions and string constants
- URL encoding and decoding functions
- HTTP status code text lookup
- HTTP date/time parsing utilities
- Cookie formatting constants

36.3 Constants, types and variables

36.3.1 Constants

```
HeaderAccept = 'Accept'
```

Accept header name string.

```
HeaderAcceptCharset = 'Accept-Charset'
```

Accept-Charset header name string.

```
HeaderAcceptEncoding = 'Accept-Encoding'
```

Accept-Encoding header name string.

```
HeaderAcceptLanguage = 'Accept-Language'
```

Accept-Language header name string.

```
HeaderAcceptRanges = 'Accept-Ranges'
```

Accept-Ranges header name string.

```
HeaderAge = 'Age'
```

Age header name string.

```
HeaderAllow = 'Allow'
```

Allow header name string.

```
HeaderAuthorization = 'Authorization'
```

Authorization header name string.

```
HeaderCacheControl = 'Cache-Control'
```

Cache-Control header name string.

```
HeaderConnection = 'Connection'
```

Connection header name string.

```
HeaderContentEncoding = 'Content-Encoding'
```

Content-Encoding header name string.

```
HeaderContentLanguage = 'Content-Language'
```

Content-Language header name string.

```
HeaderContentLength = 'Content-Length'
```

Content-Length header name string.

```
HeaderContentLocation = 'Content-Location'
```

Content-Location header name string.

```
HeaderContentMD5 = 'Content-MD5'
```

Content-MD5 header name string.

```
HeaderContentRange = 'Content-Range'
```

Content-Range header name string.

```
HeaderContentType = 'Content-Type'
```

Content-Type header name string.

```
HeaderCookie = 'Cookie'
```

String constant for the 'Cookie' HTTP header used to send cookies from client to server.

```
HeaderDate = 'Date'
```

Date header name string.

```
HeaderETag = 'ETag'
```

ETag header name string.

```
HeaderExpect = 'Expect'
```

Expect header name string.

```
HeaderExpires = 'Expires'
```

Expires header name string.

```
HeaderFrom = 'From'
```

From header name string.

```
HeaderHost = 'Host'
```

Host header name string.

```
HeaderIfMatch = 'If-Match'
```

If-Match header name string.

```
HeaderIfModifiedSince = 'If-Modified-Since'
```

If-Modified-Since header name string.

```
HeaderIfNoneMatch = 'If-None-Match'
```

If-None-Match header name string.

```
HeaderIfRange = 'If-Range'
```

If-Range header name string.

```
HeaderIfUnModifiedSince = 'If-Unmodified-Since'
```

If-Unmodified-Since header name string.

```
HeaderLastModified = 'Last-Modified'
```

Last-Modified header name string.

```
HeaderLocation = 'Location'
```

Location header name string.

```
HeaderMaxForwards = 'Max-Forwards'
```

Max-Forwards header name string.

```
HeaderPragma = 'Pragma'
```

Pragma header name string.

```
HeaderProxyAuthenticate = 'Proxy-Authenticate'
```

Proxy-Authenticate header name string.

```
HeaderProxyAuthorization = 'Proxy-Authorization'
```

Proxy-Authorization header name string.

```
HeaderRange = 'Range'
```

Range header name string.

```
HeaderReferer = 'Referer'
```

Referer header name string.

```
HeaderRetryAfter = 'Retry-After'
```

Retry-After header name string.

```
HeaderServer = 'Server'
```

Server header name string.

```
HeaderSetCookie = 'Set-Cookie'
```

String constant for the 'Set-Cookie' HTTP header used to send cookies from server to client.

```
HeaderTE = 'TE'
```

TE header name string.

```
HeaderTrailer = 'Trailer'
```

Trailer header name string.

```
HeaderTransferEncoding = 'Transfer-Encoding'
```

Transfer-Encoding header name string.

```
HeaderUpgrade = 'Upgrade'
```

Upgrade header name string.

```
HeaderUserAgent = 'User-Agent'
```

User-Agent header name string.

```
HeaderVary = 'Vary'
```

Vary header name string.

```
HeaderVia = 'Via'
```

Via header name string.

```
HeaderWarning = 'Warning'
```

Warning header name string.

```
HeaderWWWAuthenticate = 'WWW-Authenticate'
```

WWW-Authenticate header name string.

```
HeaderXRequestedWith = 'X-Requested-With'
```

X-Requested-With header name string.

```
HTTPDateFmt = '"%s", dd "%s" yyyy hh:mm:ss'
```

Format string used with FormatDateTime for creating HTTP-compliant date strings.

```
HTTPDays : Array[1..7] of string = ('Sun', 'Mon', 'Tue', 'Wed', 'Thu'
  , 'Fri', 'Sat')
```

Contains standard 3-letter day names (Sun, Mon, Tue, etc.) used in HTTP date formatting.

```
HTTPHeaderDirections : Array[THeader] of THeaderDirections = ([],
  [hdRequest], [hdRequest], [hdRequest], [hdRequest], [hdResponse]
  , [hdResponse], [hdResponse], [hdRequest], [hdRequest, hdResponse
  ], [hdRequest, hdResponse], [hdRequest, hdResponse], [hdRequest, hdResponse
  ], [hdRequest, hdResponse], [hdRequest, hdResponse], [hdRequest, hdResponse
  ], [hdRequest, hdResponse], [hdRequest, hdResponse], [hdRequest, hdResponse
  ], [hdResponse], [hdRequest, hdResponse], [hdRequest], [hdRequest
  ], [hdRequest], [hdRequest], [hdRequest], [hdRequest], [hdRequest
  ], [hdRequest], [hdRequest, hdResponse], [hdResponse], [hdRequest
  ], [hdRequest, hdResponse], [hdResponse], [hdRequest], [hdRequest
  , hdResponse], [hdRequest], [hdResponse], [hdResponse], [hdRequest
  ], [hdRequest, hdResponse], [hdRequest, hdResponse], [hdRequest, hdResponse
  ], [hdRequest], [hdRequest, hdResponse], [hdRequest, hdResponse],
  [hdRequest, hdResponse], [hdResponse])
```

Maps each THeader value to its allowed usage directions (request, response, or both). Used to validate proper header usage in HTTP communications.

```
HTTPHeaderNames : THeadersArray = ('', HeaderAccept, HeaderAcceptCharset
  , HeaderAcceptEncoding, HeaderAcceptLanguage, HeaderAcceptRanges,
  HeaderAge, HeaderAllow, HeaderAuthorization, HeaderCacheControl,
  HeaderConnection, HeaderContentEncoding, HeaderContentLanguage, HeaderContentLengt
  , HeaderContentLocation, HeaderContentMD5, HeaderContentRange, HeaderContentType
  , HeaderDate, HeaderETag, HeaderExpires, HeaderExpect, HeaderFrom
  , HeaderHost, HeaderIfMatch, HeaderIfModifiedSince, HeaderIfNoneMatch
  , HeaderIfRange, HeaderIfModifiedSince, HeaderLastModified, HeaderLocation
  , HeaderMaxForwards, HeaderPragma, HeaderProxyAuthenticate, HeaderProxyAuthorizati
  , HeaderRange, HeaderReferer, HeaderRetryAfter, HeaderServer, HeaderTE
  , HeaderTrailer, HeaderTransferEncoding, HeaderUpgrade, HeaderUserAgent
  , HeaderVary, HeaderVia, HeaderWarning, HeaderWWWAuthenticate)
```

Contains the actual HTTP header name strings indexed by THeader enumeration values. Used internally by HeaderName function.

```
HTTPMonths : Array[1..12] of string = ('Jan', 'Feb', 'Mar', 'Apr'
  , 'May', 'Jun', 'Jul', 'Aug', 'Sep', 'Oct', 'Nov', 'Dec')
```

Contains standard 3-letter month names (Jan, Feb, Mar, etc.) used in HTTP date formatting.

```
SCookieDomain = ' Domain=%s'
```

Format string for cookie domain attribute.

```
SCookieExpire = ' "Expires="' + HTTPDateFmt + ' "GMT"'
```

Format string for cookie expiration date.

```
SCookieHttpOnly = ' HttpOnly'
```

String for cookie HttpOnly attribute.

```
SCookiePath = ' Path=%s'
```

Format string for cookie path attribute.

```
SCookieSameSite = ' SameSite'
```

String for cookie SameSite attribute.

```
SCookieSecure = ' Secure'
```

String for cookie Secure attribute.

36.3.2 Types

```
THeader = (hhUnknown,hhAccept,hhAcceptCharset,hhAcceptEncoding,
  hhAcceptLanguage,hhAcceptRanges,hhAge,hhAllow,
  hhAuthorization,hhCacheControl,hhConnection,
            hhContentEncoding
  ,hhContentLanguage,hhContentLength,
            hhContentLocation,hhContentMD5
  ,hhContentRange,hhContentType,
            hhDate,hhETag,hhExpires
  ,hhExpect,hhFrom,hhHost,hhIfMatch,
            hhIfModifiedSince,hhIfNoneMatch
  ,hhIfRange,
            hhIfUnModifiedSince,hhLastModified,hhLocation
  ,hhMaxForwards,
            hhPragma,hhProxyAuthenticate,hhProxyAuthorization
  ,hhRange,
            hhReferer,hhRetryAfter,hhServer,hhTE,hhTrailer
  ,
            hhTransferEncoding,hhUpgrade,hhUserAgent,hhVary,hhVia
  ,
            hhWarning,hhWWWAuthenticate)
```

Table 36.2: Enumeration values for type THeader

Value	Explanation
hhAccept	Accept header - specifies acceptable response media types.
hhAcceptCharset	Accept-Charset header - specifies acceptable character encodings.
hhAcceptEncoding	Accept-Encoding header - specifies acceptable content encodings.
hhAcceptLanguage	Accept-Language header - specifies acceptable natural languages.
hhAcceptRanges	Accept-Ranges header - indicates server support for range requests.
hhAge	Age header - indicates response age in seconds.
hhAllow	Allow header - lists supported HTTP methods for resource.
hhAuthorization	Authorization header - contains client authentication credentials.
hhCacheControl	Cache-Control header - specifies caching directives.
hhConnection	Connection header - controls connection persistence.
hhContentEncoding	Content-Encoding header - specifies content compression method.
hhContentLanguage	Content-Language header - specifies content natural language.
hhContentLength	Content-Length header - specifies content size in bytes.
hhContentLocation	Content-Location header - specifies alternative resource location.
hhContentMD5	Content-MD5 header - contains MD5 hash of content.
hhContentRange	Content-Range header - specifies partial content range.
hhContentType	Content-Type header - specifies media type of content.
hhDate	Date header - specifies message origination date and time.
hhETag	ETag header - provides entity tag for caching validation.
hhExpect	Expect header - indicates expected server behaviors.
hhExpires	Expires header - specifies response expiration date and time.
hhFrom	From header - contains user agent email address.
hhHost	Host header - specifies target host and port number.
hhIfMatch	If-Match header - makes request conditional on entity tag.
hhIfModifiedSince	If-Modified-Since header - makes request conditional on modification date.
hhIfNoneMatch	If-None-Match header - makes request conditional on entity tag mismatch.
hhIfRange	If-Range header - makes range request conditional on entity tag or date.
hhIfUnModifiedSince	If-Unmodified-Since header - makes request conditional on lack of modification.
hhLastModified	Last-Modified header - specifies resource last modification date.
hhLocation	Location header - specifies redirect or resource location.
hhMaxForwards	Max-Forwards header - limits number of proxy forwards.
hhPragma	Pragma header - contains implementation-specific directives.
hhProxyAuthenticate	Proxy-Authenticate header - specifies proxy authentication method.
hhProxyAuthorization	Proxy-Authorization header - contains proxy authentication credentials.
hhRange	Range header - specifies partial content ranges to retrieve.
hhReferer	Referer header - specifies referring page URI.
hhRetryAfter	Retry-After header - specifies delay before retry attempt.
hhServer	Server header - specifies server software information.
hhTE	TE header - specifies acceptable transfer encodings.
hhTrailer	Trailer header - specifies header fields in chunked encoding trailer.
hhTransferEncoding	Transfer-Encoding header - specifies message body encoding method.
hhUnknown	Unknown or unrecognized header type.
hhUpgrade	Upgrade header - specifies protocol upgrade options.
hhUserAgent	User-Agent header - specifies client software information.
hhVary	Vary header - specifies headers affecting response selection.
hhVia	Via header - specifies intermediate proxy information.
hhWarning	Warning header - provides additional warning information.
hhWWWAuthenticate	WWW-Authenticate header - specifies server authentication method.

This enumeration defines all standard HTTP 1.1 headers as defined in RFC 2616, plus some commonly used non-standard headers like X-Requested-With, Cookie, and Set-Cookie.

```
THeaderDirection = (hdRequest,hdResponse)
```

Table 36.3: Enumeration values for type THeaderDirection

Value	Explanation
hdRequest	Header used in HTTP requests.
hdResponse	Header used in HTTP responses.

Indicates whether a header is used in requests, responses, or both.

```
THeaderDirections = Set of THeaderDirection
```

Set of header direction values.

```
THeaders = Set of THeader
```

A set type for working with multiple HTTP headers simultaneously.

```
THeadersArray = Array[THeader] of string
```

Array of header name strings indexed by THeader.

36.4 Procedures and functions

36.4.1 ExcludeHTTPPathDelimiter

Synopsis: Removes trailing forward slash from a URL path.

Declaration: `function ExcludeHTTPPathDelimiter(const AStr: string) : string`

Visibility: default

Description: Removes the trailing forward slash from a path string if present. Useful for normalizing file paths in URLs.

See also: IncludeHTTPPathDelimiter (1224)

36.4.2 HeaderName

Synopsis: Returns the string name of an HTTP header.

Declaration: `function HeaderName(AHeader: THeader) : string`

Visibility: default

Description: Converts a THeader (1221) enumeration value to its corresponding HTTP header string.

See also: HeaderType (1224), THeader (1221)

36.4.3 HeaderType

Synopsis: Returns the header type for a given header string.

Declaration: `function HeaderType(AHeader: string) : THeader`

Visibility: default

Description: Converts an HTTP header string to its corresponding THeader (1221) enumeration value. Returns hhUnknown if the header is not recognized.

See also: HeaderName (1223), THeader (1221)

36.4.4 HTTPDecode

Synopsis: Decodes HTTP URL-encoded strings.

Declaration: `function HTTPDecode(const AStr: string) : string`

Visibility: default

Description: Decodes percent-encoded characters (%XX) and optionally converts plus signs to spaces. The overloaded version without aPlusAsSpaces defaults to treating plus signs as spaces.

See also: HTTPEncode (1224)

36.4.5 HTTPEncode

Synopsis: Encodes strings for HTTP URL transmission.

Declaration: `function HTTPEncode(const AStr: string) : string`

Visibility: default

Description: Converts unsafe characters to percent-encoded format (%XX). The overloaded version allows specifying custom unsafe characters and space handling behavior.

See also: HTTPDecode (1224)

36.4.6 IncludeHTTPPathDelimiter

Synopsis: Ensures a URL path ends with a forward slash.

Declaration: `function IncludeHTTPPathDelimiter(const AStr: string) : string`

Visibility: default

Description: Adds a trailing forward slash to a path string if it doesn't already have one. Useful for normalizing directory paths in URLs.

See also: ExcludeHTTPPathDelimiter (1223)

Chapter 37

Reference for unit 'httproute'

37.1 Used units

Table 37.1: Used units by unit 'httproute'

Name	Page
Classes	??
HTTPDefs	1141
System	??
sysutils	??

37.2 Overview

The `HttpRoute` unit provides HTTP request routing functionality for web applications. It defines the THTTPRouter (1240) class, which allows mapping URL patterns to specific handlers based on HTTP methods and path patterns, with support for callbacks, events, objects, and interfaces as route handlers.

A global instance of the router is available in HTTPRouter (1234), this is where you must register your routes: the rest of the FPC web framework uses this global instance to determine where to route requests to.

37.3 Advanced HTTP Router Handler Examples

Beyond simple procedure callbacks, the HTTP router supports sophisticated handler patterns for complex applications. These examples demonstrate method events, interface handlers, and object class handlers.

```
// Method Event Handler Example
type
  TAPIHandler = class
  private
    FVersion: String;
  public
```

```
    constructor Create(const AVersion: String);
    procedure HandleVersionInfo(ARequest: TRequest; AResponse: TResponse);
  end;

constructor TAPIHandler.Create(const AVersion: String);
begin
  inherited Create;
  FVersion := AVersion;
end;

procedure TAPIHandler.HandleVersionInfo(ARequest: TRequest; AResponse: TResponse);
begin
  AResponse.Content := Format('{"version": "%s"}', [FVersion]);
  AResponse.ContentType := 'application/json';
end;

// Interface Handler Example
type
  IUserService = interface(IRouteInterface)
    ['{12345678-1234-1234-1234-123456789012}']
  end;

  TUserService = class(TInterfacedObject, IUserService)
  public
    procedure HandleRequest(ARequest: TRequest; AResponse: TResponse);
  end;

procedure TUserService.HandleRequest(ARequest: TRequest; AResponse: TResponse);
begin
  AResponse.Content := 'User service response';
end;

// Object Class Handler Example
type
  TRequestProcessor = class(TRouteObject)
  public
    procedure HandleRequest(ARequest: TRequest; AResponse: TResponse); override;
  end;

procedure TRequestProcessor.HandleRequest(ARequest: TRequest; AResponse: TResponse);
begin
  AResponse.Content := 'Processed by object instance';
end;

// Registration examples:
var
  APIHandler: TAPIHandler;
  UserService: IUserService;
begin
  APIHandler := TAPIHandler.Create('1.0');
  UserService := TUserService.Create;

  // Register method event
  HTTPRouter.RegisterRoute('/version', rmGet, @APIHandler.HandleVersionInfo);
```

```
  // Register interface handler
  HTTPRouter.RegisterRoute('/users', rmGet, UserService);

  // Register object class handler
  HTTPRouter.RegisterRoute('/process', rmPost, TRequestProcessor);
end.
```

These examples simply demonstrate the core functionality of the HTTP router. An actual implementation would typically integrate with an HTTP server framework like `fpweb` or `fphttp` to handle incoming requests, and would use one of the various request-handling units such as httpapp (1225) or cgiapp (1225).

37.4 HTTP Router Methods and RESTful APIs

Routes can be configured to handle different HTTP methods (GET, POST, PUT, DELETE, etc.). The same URL pattern can have different handlers for different methods, enabling RESTful API design. This allows implementing CRUD operations on the same resource endpoint.

```
program MethodRouter;

{$mode objfpc}{$H+}

uses
  httproute, httpdefs;

procedure HandleGetUsers(ARequest: TRequest; AResponse: TResponse);
begin
  AResponse.Content := '{"users": ["alice", "bob", "charlie"]}';
  AResponse.ContentType := 'application/json';
end;

procedure HandleCreateUser(ARequest: TRequest; AResponse: TResponse);
begin
  AResponse.Content := '{"status": "user created", "id": 123}';
  AResponse.ContentType := 'application/json';
  AResponse.Code := 201;
end;

procedure HandleUpdateUser(ARequest: TRequest; AResponse: TResponse);
begin
  AResponse.Content := '{"status": "user updated"}';
  AResponse.ContentType := 'application/json';
end;

procedure HandleDeleteUser(ARequest: TRequest; AResponse: TResponse);
begin
  AResponse.Code := 204; // No Content
end;

begin
  HTTPRouter.RegisterRoute('/users', rmGet, @HandleGetUsers, False);
```

```
  HTTPRouter.RegisterRoute('/users', rmPost, @HandleCreateUser, False);
  HTTPRouter.RegisterRoute('/users/:id', rmPut, @HandleUpdateUser, False);
  HTTPRouter.RegisterRoute('/users/:id', rmDelete, @HandleDeleteUser, False);

  Writeln('RESTful API routing:');
  Writeln('GET /users - List users');
  Writeln('POST /users - Create user');
  Writeln('PUT /users/:id - Update user');
  Writeln('DELETE /users/:id - Delete user');
end.
```

37.5 HTTP Router Interceptors and Middleware

Interceptors allow processing requests before and after route handling. This enables middleware functionality like authentication, logging, CORS, and request validation. Interceptors are implemented as object methods that receive the request, response, and a continuation flag.

Interceptors can run at two different times: `iaBefore` (before route processing) and `iaAfter` (after route processing). The continuation parameter allows interceptors to halt request processing if needed.

```
program MiddlewareRouter;

{$mode objfpc}{$H+}

uses
  httproute, httpdefs, SysUtils;

type
  TMiddlewareHelper = class
  public
    procedure LoggingInterceptor(ARequest: TRequest; AResponse: TResponse; var aCont
    procedure CORSInterceptor(ARequest: TRequest; AResponse: TResponse; var aContinu
  end;

procedure TMiddlewareHelper.LoggingInterceptor(ARequest: TRequest;
  AResponse: TResponse; var aContinue: Boolean);
begin
  Writeln(Format('[%s] %s %s', [FormatDateTime('yyyy-mm-dd hh:nn:ss', Now),
    RouteMethodToString(THTTPRouter.StringToRouteMethod(ARequest.Method)),
    ARequest.PathInfo]));
  aContinue := True;
end;

procedure TMiddlewareHelper.CORSInterceptor(ARequest: TRequest;
  AResponse: TResponse; var aContinue: Boolean);
begin
  AResponse.SetCustomHeader('Access-Control-Allow-Origin', '*');
  AResponse.SetCustomHeader('Access-Control-Allow-Methods', 'GET, POST, PUT, DELETE'
  aContinue := True;
end;

procedure HandleAPI(ARequest: TRequest; AResponse: TResponse);
```

```
begin
  AResponse.Content := '{"message": "API response with middleware"}';
  AResponse.ContentType := 'application/json';
end;

var
  Helper: TMiddlewareHelper;
begin
  Helper := TMiddlewareHelper.Create;
  try
    // Register interceptors (middleware)
    HTTPRouter.RegisterInterceptor('logging', @Helper.LoggingInterceptor, iaBefore);
    HTTPRouter.RegisterInterceptor('cors', @Helper.CORSInterceptor, iaAfter);

    // Register API route
    Router.RegisterRoute('/api/data', rmGet, @HandleAPI, False);

    Writeln('Middleware router with logging and CORS support');
    Writeln('GET /api/data - Returns JSON with proper headers');
  finally
    Helper.Free;
  end;
end.
```

37.6 HTTP Router Parameterized Routes

Routes can include parameters using pattern matching with colon notation. Parameters are extracted from the URL path and made available to handlers through string lists. This enables RESTful URL patterns like `/users/:id` or `/users/:uid/posts/:pid`.

```
program ParameterRouter;

{$mode objfpc}{$H+}

uses
  httproute, httpdefs, Classes, SysUtils;

procedure HandleUser(ARequest: TRequest; AResponse: TResponse);
var
  Params: TStringList;
  UserID: String;
begin
  Params := TStringList.Create;
  try
    // Extract parameters from the route
    if HTTPRouter.GetHTTPRoute(ARequest.PathInfo, rmGet, Params) <> nil then
    begin
      if Params.Count > 0 then
        UserID := Params[0]
      else
        UserID := 'unknown';
      AResponse.Content := 'User ID: ' + UserID;
```

```
    end
    else
      AResponse.Content := 'No route found';
  finally
    Params.Free;
  end;
end;

procedure HandleUserPosts(ARequest: TRequest; AResponse: TResponse);
var
  Params: TStringList;
  UserID, PostID: String;
begin
  Params := TStringList.Create;
  try
    if HTTPRouter.GetHTTPRoute(ARequest.PathInfo, rmGet, Params) <> nil then
    begin
      UserID := 'unknown';
      PostID := 'unknown';
      if Params.Count > 0 then UserID := Params[0];
      if Params.Count > 1 then PostID := Params[1];
      AResponse.Content := Format('User %s, Post %s', [UserID, PostID]);
    end
    else
      AResponse.Content := 'No route found';
  finally
    Params.Free;
  end;
end;

begin
  HTTPRouter.RegisterRoute('/user/:id', rmGet, @HandleUser, False);
  HTTPRouter.RegisterRoute('/user/:uid/post/:pid', rmGet, @HandleUserPosts, False);

  Writeln('Parameter routing examples:');
  Writeln('GET /user/123 - Shows user 123');
  Writeln('GET /user/456/post/789 - Shows post 789 for user 456');
end.
```

37.7 Basic HTTP Router Usage

The simplest way to use the HTTP router is through callback procedures. The router matches incoming requests to registered URL patterns and HTTP methods:

```
program BasicRouter;

{$mode objfpc}{$H+}

uses
  httproute, httpdefs;

procedure HandleHome(ARequest: TRequest; AResponse: TResponse);
```

```
begin
  AResponse.Content := 'Welcome to the homepage!';
end;

procedure HandleAPI(ARequest: TRequest; AResponse: TResponse);
begin
  AResponse.Content := 'API endpoint response';
end;

begin
  // Register routes
  HTTPRouter.RegisterRoute('/', rmGet, @HandleHome, False);
  HTTPRouter.RegisterRoute('/api', rmGet, @HandleAPI, False);

  Writeln('HTTP Router configured with routes:');
  Writeln('GET / - Homepage');
  Writeln('GET /api - API endpoint');
end.
```

37.8 HTTP Router Handler Types

The HTTP router supports several different types of handlers, providing flexibility in how you structure your application code. Each handler type serves different architectural needs and complexity levels.

Procedure Callbacks - Simple standalone procedures that accept `TRequest` and `TResponse` parameters. This is the most straightforward approach for basic routing needs and stateless operations.

Method Events - Object methods (`TRouteEvent`) that can access instance data and provide better organization for complex applications with state management. These handlers maintain object context between calls.

Interface Handlers - Objects implementing `IRouteInterface` provide clean separation of concerns and enable dependency injection patterns. This approach promotes testability and modularity.

Object Class Handlers - Specify a `TRouteObjectClass` that will be instantiated for each request, providing complete request isolation and automatic memory management. Each request gets a fresh instance.

Extended Callbacks - `TRouteCallBackEx` procedures that accept additional user data pointer, useful for passing configuration or context information to handlers without global variables.

Choose the handler type based on your application's complexity, state management needs, and architectural preferences. Simple applications can use procedure callbacks, while complex systems benefit from interface or object-based handlers.

37.9 HTTP Router Working Principle

The HTTP router operates on a simple principle: you register routes that map URL patterns to handler code, and when an HTTP request arrives, the router finds the best matching route and executes its handler.

The request processing workflow consists of five main steps:

1. **Route Registration**: Use `RegisterRoute` to associate URL patterns with handlers. Each route specifies a URL pattern (like `/users/:id`), an HTTP method (GET, POST, etc.), and a handler

(callback, event, interface, or object class).

2. **Request Processing**: When `RouteRequest` is called with an incoming HTTP request, the router searches through registered routes to find matches based on the request's path and HTTP method.

3. **Pattern Matching**: The router supports parameterized routes using colon notation (like `:id`). Parameters are extracted from the URL and made available to handlers through string lists.

4. **Handler Execution**: Once a matching route is found, its associated handler is executed with the HTTP request and response objects, along with any extracted parameters.

5. **Interceptor Processing**: Before and after route handling, registered interceptors can process the request and response for cross-cutting concerns like logging, authentication, and CORS.

The router prioritizes routes based on specificity, with exact matches taking precedence over parameterized routes, and parameterized routes taking precedence over default routes.

37.10 Constants, types and variables

37.10.1 Constants

```
RouteMethodNames : Array[TRouteMethod] of string = ('', '', 'GET'
  , 'POST', 'PUT', 'DELETE', 'OPTIONS', 'HEAD', 'TRACE')
```

`RouteMethodNames` is an array constant that maps `TRouteMethod` enumeration values to their string representations. It provides the standard HTTP method names used for string conversion operations.

37.10.2 Types

```
THTTPRouteClass = Class of THTTPRoute
```

`THTTPRouteClass` is a class reference type that points to `THTTPRoute` or its descendants. It enables dynamic creation of route instances.

```
THTTPRouterClass = Class of THTTPRouter
```

`THTTPRouterClass` is a class reference type that points to `THTTPRouter` or its descendants. It is used for creating router instances dynamically.

```
THTTPRouteRequestEvent = procedure(Sender: TObject; ARequest: TRequest
  ;
                                   AResponse: TResponse) of
  object
```

`THTTPRouteRequestEvent` defines an event type for handling HTTP route requests with sender context. It provides access to the sender object along with request and response objects.

```
TRouteCallBack = procedure(ARequest: TRequest; AResponse: TResponse
  )
```

`TRouteCallBack` defines a procedure type for handling HTTP requests in callback-based routes. It receives request and response objects for processing.

```
TRouteCallBackEx = procedure(AData: Pointer; ARequest: TRequest;
  AResponse: TResponse)
```

`TRouteCallBackEx` defines an extended callback procedure type that accepts additional user data. It provides a way to pass custom data to the callback handler.

```
TRouteEvent = procedure(ARequest: TRequest; AResponse: TResponse)
  of object
```

`TRouteEvent` defines an object method type for handling HTTP requests in event-based routes. It provides object-oriented request handling with sender context.

```
TRouteMethod = (rmUnknown,rmAll,rmGet,rmPost,rmPut,rmDelete,rmOptions
  ,
                rmHead,rmTrace)
```

Table 37.2: Enumeration values for type TRouteMethod

Value	Explanation
rmAll	Matches all HTTP methods
rmDelete	HTTP DELETE method
rmGet	HTTP GET method
rmHead	HTTP HEAD method
rmOptions	HTTP OPTIONS method
rmPost	HTTP POST method
rmPut	HTTP PUT method
rmTrace	HTTP TRACE method
rmUnknown	Unknown or unspecified HTTP method

`TRouteMethod` defines the HTTP methods that can be matched by routes. It includes common HTTP methods like GET, POST, PUT, DELETE, and others.

```
TRouteObjectClass = Class of TRouteObject
```

`TRouteObjectClass` is a class reference type that points to `TRouteObject` or its descendants. It enables dynamic creation of route handler object instances.

```
TRouteOption = (roCaseSensitive,roEmptyMatchesAll)
```

Table 37.3: Enumeration values for type TRouteOption

Value	Explanation
roCaseSensitive	Enable case-sensitive pattern matching
roEmptyMatchesAll	Allow empty patterns to match all requests

`TRouteOption` defines options that control how routes are matched against incoming requests. These options affect case sensitivity and empty pattern behavior.

```
TRouteOptions = Set of TRouteOption
```

`TRouteOptions` is a set type containing `TRouteOption` values. It allows combining multiple route matching options for flexible route configuration.

37.11 Procedures and functions

37.11.1 HTTPRouter

Synopsis: Returns the global HTTP router instance

Declaration: `function HTTPRouter : THTTPRouter`

Visibility: default

Description: `HTTPRouter` provides access to the global HTTP router service instance. It creates the router if it doesn't exist and returns the singleton instance.

37.11.2 RouteMethodToString

Synopsis: Converts route method enumeration to string

Declaration: `function RouteMethodToString(R: TRouteMethod) : string`

Visibility: default

Description: `RouteMethodToString` converts a `TRouteMethod` enumeration value to its string representation. It returns standard HTTP method names like "GET", "POST", etc.

37.12 EHTTPRoute

37.12.1 Description

`EHTTPRoute` is an exception class for HTTP routing errors. It extends `EHTTP` to provide specific error handling for routing operations.

See also: EHTTP (1225)

37.13 IRouteInterface

37.13.1 Description

`IRouteInterface` defines an interface for objects that can handle HTTP requests. It provides a standardized way to implement route handlers using interface-based design.

37.13.2 Method overview

Page	Method	Description
1234	HandleRequest	Handles an HTTP request

37.13.3 IRouteInterface.HandleRequest

Synopsis: Handles an HTTP request

Declaration: `procedure HandleRequest(ARequest: TRequest; AResponse: TResponse)`

Visibility: default

Description: `HandleRequest` processes an incoming HTTP request and generates an appropriate response. This method must be implemented by classes that support the `IRouteInterface` interface.

37.14 THTTPRoute

37.14.1 Description

`THTTPRoute` is the base class for all HTTP routes in the routing system. It provides pattern matching functionality and request handling infrastructure for specific route implementations.

See also: THTTPRouter (1240), TRouteMethod (1233), TRouteOptions (1233)

37.14.2 Method overview

Page	Method	Description
1235	Destroy	Destroys the HTTP route instance
1235	HandleRequest	Handles an HTTP request
1235	Matches	Checks if route matches pattern and method
1236	MatchMethod	Checks if route matches HTTP method
1236	MatchPattern	Matches URL pattern against request path

37.14.3 Property overview

Page	Properties	Access	Description
1236	Default	rw	Indicates if this is a default route
1237	Method	rw	HTTP method handled by this route
1236	URLPattern	rw	URL pattern for route matching

37.14.4 THTTPRoute.Destroy

Synopsis: Destroys the HTTP route instance

Declaration:
```
destructor Destroy;  Override
```

Visibility: public

Description: `Destroy` performs cleanup when the route instance is destroyed. It ensures proper resource cleanup for the route.

37.14.5 THTTPRoute.HandleRequest

Synopsis: Handles an HTTP request

Declaration:
```
procedure HandleRequest(ARequest: TRequest; AResponse: TResponse)
```

Visibility: public

Description: `HandleRequest` processes an incoming HTTP request and generates an appropriate response. This method delegates to `DoHandleRequest` for actual request processing.

37.14.6 THTTPRoute.Matches

Synopsis: Checks if route matches pattern and method

Declaration:
```
function Matches(const APattern: string; AMethod: TRouteMethod;
                 Options: TRouteOptions) : Boolean
```

Visibility: public

Description: `Matches` determines whether this route matches the specified URL pattern and HTTP method. It combines pattern and method matching to determine route eligibility.

37.14.7 THTTPRoute.MatchPattern

Synopsis: Matches URL pattern against request path

Declaration:
```
function MatchPattern(const Path: string; L: TStrings;
                      Options: TRouteOptions) : Boolean
```

Visibility: public

Description: `MatchPattern` performs pattern matching between the route's URL pattern and an incoming request path. It supports parameterized routes and extracts path parameters.

37.14.8 THTTPRoute.MatchMethod

Synopsis: Checks if route matches HTTP method

Declaration:
```
function MatchMethod(const AMethod: TRouteMethod) : Boolean
```

Visibility: public

Description: `MatchMethod` determines whether this route handles the specified HTTP method. It compares the route's method against the request method.

37.14.9 THTTPRoute.Default

Synopsis: Indicates if this is a default route

Declaration:
```
Property Default : Boolean
```

Visibility: published

Access: Read,Write

Description: `Default` specifies whether this route serves as the default handler for its HTTP method. Default routes are used when no other routes match the request.

37.14.10 THTTPRoute.URLPattern

Synopsis: URL pattern for route matching

Declaration:
```
Property URLPattern : string
```

Visibility: published

Access: Read,Write

Description: `URLPattern` defines the URL pattern that this route matches. It can include parameter placeholders and supports pattern-based routing.

37.14.11 THTTPRoute.Method

Synopsis: HTTP method handled by this route

Declaration: `Property Method : TRouteMethod`

Visibility: published

Access: Read,Write

Description: `Method` specifies which HTTP method this route handles. Routes can be configured to handle specific methods like GET, POST, etc.

37.15 THTTPRouteCallback

37.15.1 Description

`THTTPRouteCallback` implements a route that handles requests using a callback procedure. It executes the assigned callback when the route is matched.

37.15.2 Property overview

Page	Properties	Access	Description
1237	CallBack	rw	Callback procedure for handling requests

37.15.3 THTTPRouteCallback.CallBack

Synopsis: Callback procedure for handling requests

Declaration: `Property CallBack : TRouteCallBack`

Visibility: public

Access: Read,Write

Description: `CallBack` is the procedure that gets executed when this route handles a request. It implements the actual request processing logic.

37.16 THTTPRouteCallbackEx

37.16.1 Description

`THTTPRouteCallbackEx` implements a route that handles requests using an extended callback procedure. It supports passing custom data to the callback handler.

37.16.2 Property overview

Page	Properties	Access	Description
1238	CallBack	rw	Extended callback procedure for handling requests
1238	Data	rw	User data passed to callback

37.16.3 THTTPRouteCallbackEx.CallBack

Synopsis: Extended callback procedure for handling requests

Declaration: `Property CallBack : TRouteCallBackEx`

Visibility: public

Access: Read,Write

Description: `CallBack` is the extended callback procedure that gets executed when this route handles a request. It receives additional user data along with request and response objects.

37.16.4 THTTPRouteCallbackEx.Data

Synopsis: User data passed to callback

Declaration: `Property Data : Pointer`

Visibility: public

Access: Read,Write

Description: `Data` contains user-defined data that is passed to the callback procedure. It allows passing context or configuration data to the handler.

37.17 THTTPRouteEvent

37.17.1 Description

`THTTPRouteEvent` implements a route that handles requests using an object method event. It provides object-oriented request handling capabilities.

37.17.2 Property overview

Page	Properties	Access	Description
1238	Event	rw	Event handler for processing requests

37.17.3 THTTPRouteEvent.Event

Synopsis: Event handler for processing requests

Declaration: `Property Event : TRouteEvent`

Visibility: public

Access: Read,Write

Description: `Event` is the object method that gets executed when this route handles a request. It provides access to the sender object context during request processing.

37.18 THTTPRouteInterface

37.18.1 Description

`THTTPRouteInterface` implements a route that handles requests using an interface-based handler. It delegates request processing to an object implementing `IRouteInterface`.

37.18.2 Property overview

Page	Properties	Access	Description
1239	Intf	rw	Interface handler for processing requests

37.18.3 THTTPRouteInterface.Intf

Synopsis: Interface handler for processing requests

Declaration: `Property Intf : IRouteInterface`

Visibility: public

Access: Read,Write

Description: `Intf` is the interface object that handles requests for this route. It must implement the `IRouteInterface` interface to process HTTP requests.

37.19 THTTPRouteList

37.19.1 Description

`THTTPRouteList` maintains a collection of `THTTPRoute` objects. It provides indexed access to routes and manages the route collection.

See also: THTTPRoute (1235), THTTPRouter (1240)

37.19.2 Property overview

Page	Properties	Access	Description
1239	Routes	rw	Provides indexed access to routes

37.19.3 THTTPRouteList.Routes

Synopsis: Provides indexed access to routes

Declaration: `Property Routes[AIndex: Integer]: THTTPRoute; default`

Visibility: public

Access: Read,Write

Description: `Routes` allows accessing individual routes in the collection by index. It provides both read and write access to route objects.

37.20 THTTPRouteObject

37.20.1 Description

`THTTPRouteObject` implements a route that handles requests by creating instances of a specified object class. It instantiates the object class for each request to provide isolated request handling.

37.20.2 Property overview

Page	Properties	Access	Description
1240	ObjectCLass	rw	Object class for handling requests

37.20.3 THTTPRouteObject.ObjectCLass

Synopsis: Object class for handling requests

Declaration: `Property ObjectCLass : TRouteObjectClass`

Visibility: public

Access: Read,Write

Description: `ObjectCLass` specifies the class that will be instantiated to handle requests for this route. The class must descend from `TRouteObject` and implement the `HandleRequest` method.

37.21 THTTPRouter

37.21.1 Description

`THTTPRouter` provides HTTP request routing functionality for web applications. It manages a collection of routes and matches incoming HTTP requests to appropriate handlers based on URL patterns and HTTP methods.

37.21.2 Method overview

Page	Method	Description
1241	Create	Creates a new HTTP router instance
1241	DeleteRoute	Removes a route from the collection
1241	DeleteRouteByID	Removes a route by its ID
1241	Destroy	Destroys the HTTP router instance
1243	FindHTTPRoute	Finds a route matching path and method
1243	GetHTTPRoute	Gets the best matching route for path and method
1242	RegisterRoute	Registers a new route with the router
1244	RouteRequest	Routes an HTTP request to appropriate handler
1241	SanitizeRoute	Sanitizes a route path string
1242	Service	Returns the global router service instance
1242	ServiceClass	Returns the router service class
1242	SetServiceClass	Sets the router service class
1242	StringToRouteMethod	Converts string to route method enumeration

37.21.3 Property overview

Page	Properties	Access	Description
1245	AfterRequest	rw	Event triggered after processing requests
1244	BeforeRequest	rw	Event triggered before processing requests
1244	RouteCount	r	Returns the number of registered routes
1245	RouteOptions	rw	Global options for route matching
1244	Routes	r	Provides indexed access to routes

37.21.4 THTTPRouter.Create

Synopsis: Creates a new HTTP router instance

Declaration:
```
constructor Create(AOwner: TComponent);  Override
```

Visibility: public

Description: `Create` initializes a new HTTP router with default settings. It sets up the internal route collection and interceptor management.

37.21.5 THTTPRouter.Destroy

Synopsis: Destroys the HTTP router instance

Declaration:
```
destructor Destroy;  Override
```

Visibility: public

Description: `Destroy` performs cleanup when the router is destroyed. It releases all routes, interceptors, and associated resources.

37.21.6 THTTPRouter.DeleteRoute

Synopsis: Removes a route from the collection

Declaration:
```
procedure DeleteRoute(AIndex: Integer)
procedure DeleteRoute(ARoute: THTTPRoute)
```

Visibility: public

Description: `DeleteRoute` removes the specified route from the router's route collection. It can delete by index or by route object reference.

37.21.7 THTTPRouter.DeleteRouteByID

Synopsis: Removes a route by its ID

Declaration:
```
procedure DeleteRouteByID(AID: Integer)
```

Visibility: public

Description: `DeleteRouteByID` removes a route from the collection using its unique identifier. It searches for the route with the specified ID and removes it.

37.21.8 THTTPRouter.SanitizeRoute

Synopsis: Sanitizes a route path string

Declaration:
```
class function SanitizeRoute(const Path: string) : string
```

Visibility: public

Description: `SanitizeRoute` cleans and normalizes a route path string to ensure proper format. It removes invalid characters and applies standard formatting rules.

37.21.9 THTTPRouter.Service

Synopsis: Returns the global router service instance

Declaration: `class function Service : THTTPRouter`

Visibility: public

Description: `Service` provides access to the global router service instance. It implements a singleton pattern for router access across the application.

37.21.10 THTTPRouter.ServiceClass

Synopsis: Returns the router service class

Declaration: `class function ServiceClass : THTTPRouterClass`

Visibility: public

Description: `ServiceClass` returns the class used for creating router service instances. It allows customization of the router implementation.

37.21.11 THTTPRouter.SetServiceClass

Synopsis: Sets the router service class

Declaration: `class procedure SetServiceClass(AClass: THTTPRouterClass)`

Visibility: public

Description: `SetServiceClass` configures the class to use for creating router service instances. It allows replacing the default router implementation with a custom one.

37.21.12 THTTPRouter.StringToRouteMethod

Synopsis: Converts string to route method enumeration

Declaration: `class function StringToRouteMethod(const S: string) : TRouteMethod`

Visibility: public

Description: `StringToRouteMethod` parses a string representation of an HTTP method and returns the corresponding `TRouteMethod` enumeration value. It supports standard HTTP method names like "GET", "POST", etc.

37.21.13 THTTPRouter.RegisterRoute

Synopsis: Registers a new route with the router

Declaration:
```
function RegisterRoute(const APattern: string; AEvent: TRouteEvent;
                       IsDefault: Boolean) : THTTPRoute;  Overload
function RegisterRoute(const APattern: string; AMethod: TRouteMethod;
                       AEvent: TRouteEvent; IsDefault: Boolean)
                        : THTTPRoute;  Overload
function RegisterRoute(const APattern: string;
                       const AIntf: IRouteInterface; IsDefault: Boolean)
                        : THTTPRoute;  Overload
```

```
function RegisterRoute(const APattern: string; AMethod: TRouteMethod;
                       const AIntf: IRouteInterface; IsDefault: Boolean)
                        : THTTPRoute;  Overload
function RegisterRoute(const APattern: string;
                       const AObjectClass: TRouteObjectClass;
                       IsDefault: Boolean) : THTTPRoute;  Overload
function RegisterRoute(const APattern: string; AMethod: TRouteMethod;
                       const AobjectClass: TRouteObjectClass;
                       IsDefault: Boolean) : THTTPRoute;  Overload
function RegisterRoute(const APattern: string; AData: Pointer;
                       ACallBack: TRouteCallBackEx; IsDefault: Boolean)
                        : THTTPRoute;  Overload
function RegisterRoute(const APattern: string; AData: Pointer;
                       AMethod: TRouteMethod;
                       ACallBack: TRouteCallBackEx; IsDefault: Boolean)
                        : THTTPRoute;  Overload
function RegisterRoute(const APattern: string;
                       ACallBack: TRouteCallBack; IsDefault: Boolean)
                        : THTTPRoute;  Overload
function RegisterRoute(const APattern: string; AMethod: TRouteMethod;
                       ACallBack: TRouteCallBack; IsDefault: Boolean)
                        : THTTPRoute;  Overload
```

Visibility: public

Description: `RegisterRoute` adds a new route to the router with the specified pattern and handler. It supports various handler types including callbacks, events, interfaces, and object classes.

37.21.14 THTTPRouter.FindHTTPRoute

Synopsis: Finds a route matching path and method

Declaration:
```
function FindHTTPRoute(const Path: string; AMethod: TRouteMethod;
                       Params: TStrings; out MethodMismatch: Boolean)
                        : THTTPRoute
```

Visibility: public

Description: `FindHTTPRoute` searches for a route that matches the specified path and HTTP method. It extracts parameters and provides information about method mismatches.

37.21.15 THTTPRouter.GetHTTPRoute

Synopsis: Gets the best matching route for path and method

Declaration:
```
function GetHTTPRoute(const Path: string; AMethod: TRouteMethod;
                      Params: TStrings) : THTTPRoute
```

Visibility: public

Description: `GetHTTPRoute` finds the best route that matches the specified path and method. It prioritizes exact matches over default routes and extracts path parameters.

37.21.16 THTTPRouter.RouteRequest

Synopsis: Routes an HTTP request to appropriate handler

Declaration: `procedure RouteRequest(ARequest: TRequest; AResponse: TResponse)`

Visibility: public

Description: `RouteRequest` processes an incoming HTTP request by finding the appropriate route and executing its handler. It includes interceptor processing and error handling for the complete request lifecycle.

37.21.17 THTTPRouter.Routes

Synopsis: Provides indexed access to routes

Declaration: `Property Routes[AIndex: Integer]: THTTPRoute; default`

Visibility: public

Access: Read

Description: `Routes` allows accessing individual routes in the router's collection by index. It provides read-only access to route objects for inspection and management.

37.21.18 THTTPRouter.RouteCount

Synopsis: Returns the number of registered routes

Declaration: `Property RouteCount : Integer`

Visibility: public

Access: Read

Description: `RouteCount` provides the total count of routes currently registered with the router. This includes all route types and can be used for iteration and management purposes.

37.21.19 THTTPRouter.BeforeRequest

Synopsis: Event triggered before processing requests

Declaration: `Property BeforeRequest : THTTPRouteRequestEvent`

Visibility: public

Access: Read,Write

Description: `BeforeRequest` is an event that fires before any request processing begins. It allows global pre-processing of requests before route matching and handling.

37.21.20 THTTPRouter.AfterRequest

Synopsis: Event triggered after processing requests

Declaration: `Property AfterRequest : THTTPRouteRequestEvent`

Visibility: public

Access: Read,Write

Description: `AfterRequest` is an event that fires after request processing completes. It allows global post-processing of requests and responses.

37.21.21 THTTPRouter.RouteOptions

Synopsis: Global options for route matching

Declaration: `Property RouteOptions : TRouteOptions`

Visibility: public

Access: Read,Write

Description: `RouteOptions` specifies global options that affect how all routes in the router perform pattern matching. These options control case sensitivity and empty pattern behavior.

37.22 TRouteObject

37.22.1 Description

`TRouteObject` is an abstract base class for objects that can handle HTTP requests. It implements `IRouteInterface` and serves as a foundation for custom route handler classes.

37.22.2 Interfaces overview

Page	Interfaces	Description
1234	IRouteInterface	Interface for route request handlers

37.22.3 Method overview

Page	Method	Description
1245	Create	Creates a route object instance
1246	HandleRequest	Handles an HTTP request

37.22.4 TRouteObject.Create

Synopsis: Creates a route object instance

Declaration: `constructor Create;  Virtual`

Visibility: public

Description: `Create` initializes a new route object instance. This constructor can be overridden in descendant classes for custom initialization.

37.22.5 TRouteObject.HandleRequest

Synopsis: Handles an HTTP request

Declaration:
```
procedure HandleRequest(ARequest: TRequest; AResponse: TResponse)
                        ; Virtual; Abstract
```

Visibility: public

Description: `HandleRequest` is an abstract method that must be implemented by descendant classes. It processes HTTP requests and generates appropriate responses.

Chapter 38

Reference for unit 'IBConnection'

38.1 Used units

Table 38.1: Used units by unit 'IBConnection'

Name	Page
BufDataset	213
Classes	??
DB	438
dbconst	??
ibase60dyn	??
SQLDB	1487
System	??
sysutils	??

38.2 Constants, types and variables

38.2.1 Constants

```
DEFDIALECT = 3
```

Default dialect that will be used when connecting to databases. See TIBConnection.Dialect (1252) for more details on dialects.

```
MAXBLOBSEGMENTSIZE = 65535
```

Maximum size to use when fetching blob segments.

38.2.2 Types

```
TStatusVector = Array[0..19] of ISC_STATUS
```

`TStatusVector` is the (opaque) type of the Interbase internal status vector.

38.3 TDatabaseInfo

```
TDatabaseInfo = record
  Dialect : Integer;
  ODSMajorVersion : Integer
  ;
  ServerVersion : string;
  ServerVersionString : string;
end
```

`TDatabaseInfo` is used internally by `TIBconnection` to store information about the connected database.

38.4 EIBDatabaseError

38.4.1 Description

Firebird/Interbase database error, a descendant of db.EDatabaseError (1247).

See also: db.EDatabaseError (1247)

38.4.2 Property overview

Page	Properties	Access	Description
1248	GDSErrorCode	r	Firebird/Interbase GDS error code.
1248	StatusVector	rw	Low-level IB API status vector.

38.4.3 EIBDatabaseError.StatusVector

Synopsis: Low-level IB API status vector.

Declaration: `Property StatusVector : TStatusVector`

Visibility: public

Access: Read,Write

Description: `StatusVector` contains the low-level status information returned by the last firebird/interbase APO call.

38.4.4 EIBDatabaseError.GDSErrorCode

Synopsis: Firebird/Interbase GDS error code.

Declaration: `Property GDSErrorCode : Integer; deprecated;`

Visibility: public

Access: Read

Description: Firebird/Interbase-specific error code, the GDS error code. From a Firebird perspective: Firebird throws 2 error codes for an exception. The high-level one is the SQLCODE, which is a negative 3-digit code. The lower-level one is the ISC code (or GDSCODE) which has 9 digits. Related ISC error types are grouped under the same SQLCODE. In some cases, each successive gdscode error gives you further information about the error condition. Note: SQLCODE is a deprecated SQL standard; its successor is SQLSTATE.

38.5 TIBConnection

38.5.1 Description

`TIBConnection` is a descendant of TSQLConnection (1247) and represents a connection to a Firebird/Interbase server.

It is designed to work with Interbase 6, Firebird 1 and newer database servers.

`TIBConnection` by default requires the Firebird/Interbase client library (e.g. gds32.dll, libfbclient.so, fbclient.dll,fbembed.dll) and its dependencies to be installed on the system. The bitness between library and your application must match: e.g. use 32 bit fbclient when developing a 32 bit application on 64 bit Linux.

On Windows, in accordance with the regular Windows way of loading DLLs, the library can also be in the executable directory. In fact, this directory is searched first, and might be a good option for distributing software to end users as it eliminates problems with incompatible DLL versions.

`TIBConnection` is based on FPC Interbase/Firebird code (ibase60.inc) that tries to load the client library. If you want to use Firebird embedded, make sure the embedded library is searched/loaded first. There are several ways to do this:

- Include `ibase60` in your uses clause, set `UseEmbeddedFirebird` to true
- On Windows, with FPC newer than 2.5.1, put fbembed.dll in your application directory
- On Windows, put the fbembed.dll in your application directory and rename it to fbclient.dll

Pre 2.5.1 versions of FPC did not try to load the fbembed library by default. See FPC bug 17664 for more details.

An indication of which DLLs need to be installed on Windows (Firebird 2.5, differs between versions:

- fbclient.dll (or fbembed.dll)
- firebird.msg
- ib_util.dll
- icudt30.dll
- icuin30.dll
- icuuc30.dll
- msvcp80.dll
- msvcr80.dll

Please see your database documentation for details.

The `TIBConnection` component does not reliably detect computed fields as such. This means that automatically generated update SQL statements will attempt to update these fields, resulting in SQL errors. These errors can be avoided by removing the `pfInUpdate` flag from the provideroptions from a field, once it has been created:

```
MyQuery.FieldByName('full_name').ProviderFlags:=[];
```

See also: TSQLConnection (1247)

38.5.2 Method overview

Page	Method	Description
1250	Create	Creates a `TIBConnection` object.
1251	CreateDB	Creates a database on disk.
1251	DropDB	Deletes a database from disk.
1250	GetConnectionInfo	Return some information about the connection.

38.5.3 Property overview

Page	Properties	Access	Description
1251	BlobSegmentSize	rw	Write this amount of bytes per BLOB segment.
1253	CheckTransactionParams	rw	Let StartTransaction check transaction parameters.
1252	DatabaseName		Name of the database to connect to.
1252	Dialect	rws	Database dialect.
1253	KeepConnection		Keep open connection after first query.
1253	LoginPrompt		Switch for showing custom login prompt.
1252	ODSMajorVersion	r	Database On-Disk Structure major version.
1254	OnLogin		Event triggered when a login prompt needs to be shown.
1253	Params		Firebird/Interbase specific parameters.
1254	Port	s	Port at which the server listens.
1254	UseConnectionCharSetIfNone	rw	For string/blob fields with codepage none, use the connection character set when copying data.
1254	WireCompression	rw	Use wire compression when communicating with the server.

38.5.4 TIBConnection.Create

Synopsis: Creates a `TIBConnection` object.

Declaration:
```
constructor Create(AOwner: TComponent);  Override
```

Visibility: public

Description: Creates a `TIBConnection` object.

38.5.5 TIBConnection.GetConnectionInfo

Synopsis: Return some information about the connection.

Declaration:
```
function GetConnectionInfo(InfoType: TConnInfoType) : string;  Override
```

Visibility: public

Description: `GetConnectionInfo` overrides TSQLConnection.GetConnectionInfo (1518) to return the relevant information for the Interbase/Firebird connection.

See also: TSQLConnection.GetConnectionInfo (1518), TConnInfoType (1497)

38.5.6 TIBConnection.CreateDB

Synopsis: Creates a database on disk.

Declaration: `procedure CreateDB; Override`

Visibility: public

Description: Instructs the Interbase or Firebird database server to create a new database.

If set, the TSQLConnection.Params (1247) (specifically, `PAGE_SIZE`) and TSQLConnection.CharSet (1247) properties influence the database creation.

If creating a database using a client/server environment, the `TIBConnection` code will connect to the database server before trying to create the database. Therefore make sure the connection properties are already correctly set, e.g. TSQLConnection.HostName (1247), TSQLConnection.UserName (1247), TSQLConnection.Password (1247).

If creating a database using Firebird embedded, make sure the embedded library is loaded, the TSQLConnection.HostName (1247) property is empty, and set the TSQLConnection.UserName (1247) to e.g. 'SYSDBA'. See `TIBConnection`: Firebird/Interbase specific TSQLConnection (1247). for details on loading the embedded database library.

See also: TSQLConnection.Params (1247), TSQLConnection.DropDB (1247), TIBConnection (1249)

38.5.7 TIBConnection.DropDB

Synopsis: Deletes a database from disk.

Declaration: `procedure DropDB; Override`

Visibility: public

Description: `DropDB` instructs the Interbase/Firebird database server to delete the database that is specified in the TIBConnection (1249).

In a client/server environment, the `TIBConnection` code will connect to the database server before telling it to drop the database. Therefore make sure the connection properties are already correctly set, e.g. TSQLConnection.HostName (1247), TSQLConnection.UserName (1247), TSQLConnection.Password (1247).

When using Firebird embedded, make sure the embedded connection library is loaded, the TSQLConnection.HostName (1247) property is empty, and set the TSQLConnection.UserName (1247) to e.g. 'SYSDBA'. See `TIBConnection`: Firebird/Interbase specific TSQLConnection (1247). for more details on loading the embedded library.

See also: TSQLConnection.CreateDB (1247), TSQLConnection.HostName (1247), TSQLConnection.UserName (1247), TSQLConnection.Password (1247)

38.5.8 TIBConnection.BlobSegmentSize

Synopsis: Write this amount of bytes per BLOB segment.

Declaration: `Property BlobSegmentSize : Word; deprecated;`

Visibility: public

Access: Read,Write

Description: **Deprecated** since FPC 2.7.1 revision 19659

When sending BLOBs to the database, the code writes them in segments.

Before FPC 2.7.1 revision 19659, these segments were 80 bytes and could be changed using `BlobSegmentSize`. Please set BlobSegmentSize to 65535 for better write performance.

In newer FPC versions, the BlobSegmentSize property is ignored and segments of 65535 bytes are always used.

38.5.9 TIBConnection.ODSMajorVersion

Synopsis: Database On-Disk Structure major version.

Declaration: `Property ODSMajorVersion : Integer`

Visibility: public

Access: Read

Description: `ODSMajorVersion` is the Database On-Disk Structure major version. It is provided for information purposes only.

38.5.10 TIBConnection.DatabaseName

Synopsis: Name of the database to connect to.

Declaration: `Property DatabaseName :`

Visibility: published

Access:

Description: Name of the Interbase/Firebird database to connect to.

This can be either the path to the database or an alias name. Please see your database documentation for details.

In a client/server environment, the name indicates the location of the database on the server's file system, so if you have a Linux Firebird server, you might have something like /var/lib/firebird/2.5/data/employee.fdb

If using an embedded Firebird database, the name is a relative path relative to the fbembed library.

Note that the path is specified as an Ansistring, meaning that databases residing in directories that rely on Unicode characters will not work. (firebird itself also cannot handle this).

38.5.11 TIBConnection.Dialect

Synopsis: Database dialect.

Declaration: `Property Dialect : Integer`

Visibility: published

Access: Read,Write

Description: Firebird/Interbase servers since Interbase 6 have a dialect setting for backwards compatibility. It can be 1, 2 or 3, the default is 3.

Note: the dialect for new Interbase/Firebird databases is 3; dialects 1 and 2 are only used in legacy environments. In practice, you can ignore this setting for newly created databases.

38.5.12 TIBConnection.CheckTransactionParams

Synopsis: Let StartTransaction check transaction parameters.

Declaration: `Property CheckTransactionParams : Boolean`

Visibility: published

Access: Read,Write

Description: `CheckTransactionParams` can be set to `True` to force the connection component to check the transaction parameters for valid values before starting a transaction.

38.5.13 TIBConnection.KeepConnection

Synopsis: Keep open connection after first query.

Declaration: `Property KeepConnection :`

Visibility: published

Access:

Description: Determines whether to keep the connection open once it is established and the first query has been executed.

38.5.14 TIBConnection.LoginPrompt

Synopsis: Switch for showing custom login prompt.

Declaration: `Property LoginPrompt :`

Visibility: published

Access:

Description: If true, the OnLogin (1247) event will fire, allowing you to handle supplying of credentials yourself.

See also: TSQLConnection.OnLogin (1247)

38.5.15 TIBConnection.Params

Synopsis: Firebird/Interbase specific parameters.

Declaration: `Property Params :`

Visibility: published

Access:

Description: `Params` is a #rtl.classes.TStringList (**??**) of name=value combinations that set database-specific parameters.

The following parameter is supported:

- `PAGE_SIZE`: size of database pages (an integer), e.g. 16384.

See your database documentation for more details.

See also: #fcl.sqldb.TSQLConnection.Params (1525)

38.5.16 TIBConnection.OnLogin

Synopsis: Event triggered when a login prompt needs to be shown.

Declaration: `Property OnLogin :`

Visibility: published

Access:

Description: OnLogin is triggered when the connection needs a login prompt when connecting: it is triggered when the LoginPrompt (1247) property is True, after the BeforeConnect (486) event, but before the connection is actually established.

See also: #fcl.db.TCustomConnection.BeforeConnect (486), TSQLConnection.LoginPrompt (1247), #fcl.db.TCustomConnection.Open (483), TSQLConnection.OnLogin (1247)

38.5.17 TIBConnection.Port

Synopsis: Port at which the server listens.

Declaration: `Property Port :`

Visibility: published

Access:

Description: `Port` can be set to the port that Firebird is listening on. If not specified, the default port of 3050 is used when establishing a connection. This property must be set prior to activaiting the connection.

38.5.18 TIBConnection.UseConnectionCharSetIfNone

Synopsis: For string/blob fields with codepage none, use the connection character set when copying data.

Declaration: `Property UseConnectionCharSetIfNone : Boolean`

Visibility: published

Access: Read,Write

Description: `UseConnectionCharSetIfNone` can be set to true to assume that fields which have no codepage set in the database schema, use the connection character set.

See also: TSQLConnection.Charset (1522)

38.5.19 TIBConnection.WireCompression

Synopsis: Use wire compression when communicating with the server.

Declaration: `Property WireCompression : Boolean`

Visibility: published

Access: Read,Write

Description: `WireCompression` can be set to `True` to force the client to use compression when communicating with the server. This property must be set prior to activaiting the connection.

38.6 TIBConnectionDef

38.6.1 Description

Child of TConnectionDef (1247) used to register an Interbase/Firebird connection, so that it is available in "connection factory" scenarios where database drivers/connections are loaded at runtime and it is unknown at compile time whether the required database libraries are present on the end user's system.

See also: TConnectionDef (1247)

38.6.2 Method overview

Page	Method	Description
1255	ConnectionClass	Firebird/Interbase child of ConnectionClass (1504).
1256	DefaultLibraryName	Default name of the firebird client library.
1255	Description	Description for the Firebird/Interbase child of #fcl.sqldb.TConnectionDef.ConnectionClass (1504).
1256	LoadedLibraryName	Actally loaded library name.
1256	LoadFunction	Return Function to call when loading firebird support.
1255	TypeName	Firebird/Interbase child of TConnectionDef.TypeName (1247).
1256	UnLoadFunction	Return Function to call when unloading firebird support.

38.6.3 TIBConnectionDef.TypeName

Synopsis: Firebird/Interbase child of TConnectionDef.TypeName (1247).

Declaration: `class function TypeName : string;  Override`

Visibility: default

See also: TConnectionDef.TypeName (1247), TIBConnection (1249)

38.6.4 TIBConnectionDef.ConnectionClass

Synopsis: Firebird/Interbase child of ConnectionClass (1504).

Declaration: `class function ConnectionClass : TSQLConnectionClass;  Override`

Visibility: default

See also: TConnectionDef.ConnectionClass (1247), TIBConnection (1249)

38.6.5 TIBConnectionDef.Description

Synopsis: Description for the Firebird/Interbase child of #fcl.sqldb.TConnectionDef.ConnectionClass (1504).

Declaration: `class function Description : string;  Override`

Visibility: default

Description: The description identifies this `ConnectionDef` object as a Firebird/Interbase connection.

See also: #fcl.sqldb.TConnectionDef.Description (1504), TIBConnection (1249)

38.6.6 TIBConnectionDef.DefaultLibraryName

Synopsis: Default name of the firebird client library.

Declaration: `class function DefaultLibraryName : string;  Override`

Visibility: default

Description: `DefaultLibraryName` returns the library name to use when loading the firebird client library.

38.6.7 TIBConnectionDef.LoadFunction

Synopsis: Return Function to call when loading firebird support.

Declaration: `class function LoadFunction : TLibraryLoadFunction;  Override`

Visibility: default

Description: `LoadFunction` is used by the connector logic to get the function to dynamically load firebird support.

38.6.8 TIBConnectionDef.UnLoadFunction

Synopsis: Return Function to call when unloading firebird support.

Declaration: `class function UnLoadFunction : TLibraryUnLoadFunction;  Override`

Visibility: default

Description: `UnLoadFunction` is used by the connector logic to get the function to unload firebird support.

38.6.9 TIBConnectionDef.LoadedLibraryName

Synopsis: Actally loaded library name.

Declaration: `class function LoadedLibraryName : string;  Override`

Visibility: default

Description: `LoadedLibraryName` returns the actually loaded library name.

See also: DefaultLibraryName (1256)

38.7 TIBCursor

38.7.1 Description

A cursor that keeps track of where you are in a Firebird/Interbase dataset. It is a descendent of TSQLCursor (1247).

See also: TSQLCursor (1247), TIBConnection (1249)

38.8 TIBTrans

38.8.1 Description

Firebird/Interbase database transaction object. Descendant of TSQLHandle (1247).

See also: TSQLHandle (1247), TIBConnection (1249)

Chapter 39

Reference for unit 'idea'

39.1 Used units

Table 39.1: Used units by unit 'idea'

Name	Page
Classes	**??**
System	**??**
sysutils	**??**

39.2 Overview

Besides some low level IDEA encryption routines, the IDEA unit also offers 2 streams which offer on-the-fly encryption or decryption: there are 2 stream objects: A write-only encryption stream which encrypts anything that is written to it, and a decryption stream which decrypts anything that is read from it.

39.3 Constants, types and variables

39.3.1 Constants

```
IDEABLOCKSIZE = 8
```

IDEA block size.

```
IDEAKEYSIZE = 16
```

IDEA Key size constant.

```
KEYLEN = 6 * ROUNDS + 4
```

Key length.

```
ROUNDS = 8
```

Number of rounds to encrypt.

39.3.2 Types

```
IdeaCryptData = TIdeaCryptData
```

Provided for backward functionality.

```
IdeaCryptKey = TIdeaCryptKey
```

Provided for backward functionality.

```
IDEAkey = TIDEAKey
```

Provided for backward functionality.

```
TIdeaCryptData = Array[0..3] of Word
```

`TIdeaCryptData` is an internal type, defined to hold data for encryption/decryption.

```
TIdeaCryptKey = Array[0..7] of Word
```

The actual encryption or decryption key for IDEA is 64-bit long. This type is used to hold such a key. It can be generated with the EnKeyIDEA (1260) or DeKeyIDEA (1259) algorithms depending on whether an encryption or decryption key is needed.

```
TIDEAKey = Array[0..keylen-1] of Word
```

The IDEA key should be filled by the user with some random data (say, a passphrase). This key is used to generate the actual encryption/decryption keys.

39.4 Procedures and functions

39.4.1 CipherIdea

Synopsis: Encrypt or decrypt a buffer.

Declaration:
```
procedure CipherIdea(Input: TIdeaCryptData;
                     out outdata: TIdeaCryptData; z: TIDEAKey)
```

Visibility: default

Description: `CipherIdea` encrypts or decrypts a buffer with data (`Input`) using key `z`. The resulting encrypted or decrypted data is returned in `Output`.

Errors: None.

See also: EnKeyIdea (1260), DeKeyIdea (1259), TIDEAEncryptStream (1262), TIDEADecryptStream (1260)

39.4.2 DeKeyIdea

Synopsis: Create a decryption key from an encryption key.

Declaration:
```
procedure DeKeyIdea(z: TIDEAKey; out dk: TIDEAKey)
```

Visibility: default

Description: `DeKeyIdea` creates a decryption key based on the encryption key `z`. The decryption key is returned in `dk`. Note that only a decryption key generated from the encryption key that was used to encrypt the data can be used to decrypt the data.

Errors: None.

See also: EnKeyIdea (1260), CipherIdea (1259)

39.4.3 EnKeyIdea

Synopsis: Create an IDEA encryption key from a user key.

Declaration: `procedure EnKeyIdea(UserKey: TIdeaCryptKey; out z: TIDEAKey)`

Visibility: default

Description: `EnKeyIdea` creates an IDEA encryption key from user-supplied data in `UserKey`. The Encryption key is stored in `z`.

Errors: None.

See also: DeKeyIdea (1259), CipherIdea (1259)

39.5 EIDEAError

39.5.1 Description

`EIDEAError` is used to signal errors in the IDEA encryption decryption streams.

39.6 TIDEADeCryptStream

39.6.1 Description

`TIDEADecryptStream` is a stream which decrypts anything that is read from it using the IDEA mechanism. It reads the encrypted data from a source stream and decrypts it using the CipherIDEA (1259) algorithm. It is a read-only stream: it is not possible to write data to this stream.

When creating a `TIDEADecryptStream` instance, an IDEA decryption key should be passed to the constructor, as well as the stream from which encrypted data should be read written.

The encrypted data can be created with a TIDEAEncryptStream (1262) encryption stream.

See also: TIDEAEncryptStream (1262), TIDEAStream.Create (1264), CipherIDEA (1259)

39.6.2 Method overview

Page	Method	Description
1261	Create	Constructor to create a new `TIDEADecryptStream` instance.
1261	Read	Reads data from the stream, decrypting it as needed.
1261	Seek	Set position on the stream.

39.6.3 TIDEADeCryptStream.Create

Synopsis: Constructor to create a new `TIDEADecryptStream` instance.

Declaration:
```
constructor Create(const AKey: string; Dest: TStream);  Overload
```

Visibility: public

Description: `Create` creates a new `TIDEADecryptStream` instance using the string `AKey` to compute the encryption key (1259), which is then passed on to the inherited constructor TIDEAStream.Create (1264). It is an easy-access function which introduces no new functionality.

The string is truncated at the maximum length of the TIdeaCryptKey (1259) structure, so it makes no sense to provide a string with length longer than this structure.

See also: TIdeaCryptKey (1259), TIDEAStream.Create (1264), TIDEAEnCryptStream.Create (1262)

39.6.4 TIDEADeCryptStream.Read

Synopsis: Reads data from the stream, decrypting it as needed.

Declaration:
```
function Read(var Buffer; Count: LongInt) : LongInt;  Override
```

Visibility: public

Description: `Read` attempts to read `Count` bytes from the stream, placing them in `Buffer` the bytes are read from the source stream and decrypted as they are read. (bytes are read from the source stream in blocks of 8 bytes. The function returns the number of bytes actually read.

Errors: If an error occurs when reading data from the source stream, an exception may be raised.

See also: Seek (1261), TIDEAEncryptStream (1262)

39.6.5 TIDEADeCryptStream.Seek

Synopsis: Set position on the stream.

Declaration:
```
function Seek(const Offset: Int64; Origin: TSeekOrigin) : Int64
                  ;  Override
```

Visibility: public

Description: `Seek` will only work on a forward seek. It emulates a forward seek by reading and discarding bytes from the input stream. The `TIDEADecryptStream` stream tries to provide seek capabilities for the following limited number of cases:

Origin=soFromBeginning If `Offset` is larger than the current position, then the remaining bytes are skipped by reading them from the stream and discarding them.

Origin=soFromCurrent If `Offset` is zero, the current position is returned. If it is positive, then `Offset` bytes are skipped by reading them from the stream and discarding them.

Errors: An EIDEAError (1260) exception is raised if the stream does not allow the requested seek operation.

See also: Read (1261)

39.7 TIDEAEncryptStream

39.7.1 Description

`TIDEAEncryptStream` is a stream which encrypts anything that is written to it using the IDEA mechanism, and then writes the encrypted data to the destination stream using the CipherIDEA (1259) algorithm. It is a write-only stream: it is not possible to read data from this stream.

When creating a `TIDEAEncryptStream` instance, an IDEA encryption key should be passed to the constructor, as well as the stream to which encrypted data should be written.

The resulting encrypted data can be read again with a TIDEADecryptStream (1260) decryption stream.

See also: TIDEADecryptStream (1260), TIDEAStream.Create (1264), CipherIDEA (1259)

39.7.2 Method overview

Page	Method	Description
1262	Create	Constructor to create a new `TIDEAEncryptStream` instance.
1262	Destroy	Flush data buffers and free the stream instance.
1263	Flush	Write remaining bytes from the stream.
1263	Seek	Set stream position.
1263	Write	Write bytes to the stream to be encrypted.

39.7.3 TIDEAEncryptStream.Create

Synopsis: Constructor to create a new `TIDEAEncryptStream` instance.

Declaration:
```
constructor Create(const AKey: string; Dest: TStream);  Overload
```

Visibility: public

Description: `Create` creates a new `TIDEAEncryptStream` instance using the string `AKey` to compute the encryption key (1259), which is then passed on to the inherited constructor TIDEAStream.Create (1264). It is an easy-access function which introduces no new functionality.

The string is truncated at the maximum length of the TIdeaCryptKey (1259) structure, so it makes no sense to provide a string with length longer than this structure.

See also: TIdeaCryptKey (1259), TIDEAStream.Create (1264), TIDEADeCryptStream.Create (1261)

39.7.4 TIDEAEncryptStream.Destroy

Synopsis: Flush data buffers and free the stream instance.

Declaration:
```
destructor Destroy;  Override
```

Visibility: public

Description: `Destroy` flushes any data still remaining in the internal encryption buffer, and then calls the inherited `Destroy`

By default, the destination stream is not freed when the encryption stream is freed.

Errors: None.

See also: TIDEAStream.Create (1264)

39.7.5 TIDEAEncryptStream.Write

Synopsis: Write bytes to the stream to be encrypted.

Declaration: `function Write(const Buffer; Count: LongInt) : LongInt;  Override`

Visibility: public

Description: `Write` writes `Count` bytes from `Buffer` to the stream, encrypting the bytes as they are written (encryption in blocks of 8 bytes).

Errors: If an error occurs writing to the destination stream, an error may occur.

See also: Read (1261)

39.7.6 TIDEAEncryptStream.Seek

Synopsis: Set stream position.

Declaration: `function Seek(Offset: LongInt; Origin: Word) : LongInt;  Override`

Visibility: public

Description: `Seek` return the current position if called with `0` and `soFromCurrent` as arguments. With all other values, it will always raise an exception, since it is impossible to set the position on an encryption stream.

Errors: An EIDEAError (1260) will be raised unless called with `0` and `soFromCurrent` as arguments.

See also: Write (1263), EIDEAError (1260)

39.7.7 TIDEAEncryptStream.Flush

Synopsis: Write remaining bytes from the stream.

Declaration: `procedure Flush`

Visibility: public

Description: `Flush` writes the current encryption buffer to the stream. Encryption always happens in blocks of 8 bytes, so if the buffer is not completely filled at the end of the writing operations, it must be flushed. It should never be called directly, unless at the end of all writing operations. It is called automatically when the stream is destroyed.

Errors: None.

See also: Write (1263)

39.8 TIDEAStream

39.8.1 Description

Do not create instances of `TIDEAStream` directly. It implements no useful functionality: it serves as a common ancestor of the TIDEAEncryptStream (1262) and TIDEADeCryptStream (1260), and simply provides some fields that these descendent classes use when encrypting/decrypting. One of these classes should be created, depending on whether one wishes to encrypt or to decrypt.

See also: TIDEAEncryptStream (1262), TIDEADeCryptStream (1260)

39.8.2 Method overview

Page	Method	Description
1264	Create	Creates a new instance of the `TIDEAStream` class.

39.8.3 Property overview

Page	Properties	Access	Description
1264	Key	r	Key used when encrypting/decrypting.

39.8.4 TIDEAStream.Create

Synopsis: Creates a new instance of the `TIDEAStream` class.

Declaration:
```
constructor Create(AKey: TIDEAKey; Dest: TStream);  Overload
```

Visibility: public

Description: `Create` stores the encryption/decryption key and then calls the inherited `Create` to store the `Dest` stream.

Errors: None.

See also: TIDEAEncryptStream (1262), TIDEADeCryptStream (1260)

39.8.5 TIDEAStream.Key

Synopsis: Key used when encrypting/decrypting.

Declaration:
```
Property Key : TIDEAKey
```

Visibility: public

Access: Read

Description: `Key` is the key as it was passed to the constructor of the stream. It cannot be changed while data is read or written. It is the key as it is used when encrypting/decrypting.

See also: CipherIdea (1259)

Chapter 40

Reference for unit 'inicol'

40.1 Used units

Table 40.1: Used units by unit 'inicol'

Name	Page
Classes	??
IniFiles	1275
System	??
sysutils	??

40.2 Overview

inicol contains an implementation of `TCollection` and `TCollectionItem` descendents which cooperate to read and write the collection from and to a .ini file. It uses the TCustomIniFile (1277) class for this.

40.3 Constants, types and variables

40.3.1 Constants

```
KeyCount = 'Count'
```

`KeyCount` is used as a key name when reading or writing the number of items in the collection from the global section.

```
SGlobal = 'Global'
```

`SGlobal` is used as the default name of the global section when reading or writing the collection.

40.4 EIniCol

40.4.1 Description

`EIniCol` is used to report error conditions in the load and save methods of TIniCollection (1266).

40.5 TIniCollection

40.5.1 Description

`TIniCollection` is a collection (**??**) descendent which has the capability to write itself to an .ini file. It introduces some load and save mechanisms, which can be used to write all items in the collection to disk. The items should be descendents of the type TIniCollectionItem (1270).

All methods work using a `TCustomInifile` class, making it possible to save to alternate file formats, or even databases.

An instance of `TIniCollection` should never be used directly. Instead, a descendent should be used, which sets the `FPrefix` and `FSectionPrefix` protected variables.

See also: TIniCollection.LoadFromFile (1268), TIniCollection.LoadFromIni (1268), TIniCollection.SaveToIni (1267), TIniCollection.SaveToFile (1267)

40.5.2 Method overview

Page	Method	Description
1266	Load	Loads the collection from the default filename.
1268	LoadFromFile	Load collection from file.
1268	LoadFromIni	Load collection from a file in .ini file format.
1267	Save	Save the collection to the default filename.
1267	SaveToFile	Save collection to a file in .ini file format.
1267	SaveToIni	Save the collection to a `TCustomIniFile` descendent.

40.5.3 Property overview

Page	Properties	Access	Description
1269	FileName	rw	Filename of the collection.
1269	GlobalSection	rw	Name of the global section.
1268	Prefix	r	Prefix used in global section.
1269	SectionPrefix	r	Prefix string for section names.

40.5.4 TIniCollection.Load

Synopsis: Loads the collection from the default filename.

Declaration: `procedure Load`

Visibility: public

Description: `Load` loads the collection from the file as specified in the FileName (1269) property. It calls the LoadFromFile (1268) method to do this.

Errors: If the collection was not loaded or saved to file before this call, an `EIniCol` exception will be raised.

See also: TIniCollection.LoadFromFile (1268), TIniCollection.LoadFromIni (1268), TIniCollection.Save (1267), FileName (1269)

40.5.5 TIniCollection.Save

Synopsis: Save the collection to the default filename.

Declaration: `procedure Save`

Visibility: public

Description: `Save` writes the collection to the file as specified in the FileName (1269) property, using GlobalSection (1269) as the section. It calls the SaveToFile (1267) method to do this.

Errors: If the collection was not loaded or saved to file before this call, an `EIniCol` exception will be raised.

See also: TIniCollection.SaveToFile (1267), TIniCollection.SaveToIni (1267), TIniCollection.Load (1266), FileName (1269)

40.5.6 TIniCollection.SaveToIni

Synopsis: Save the collection to a `TCustomIniFile` descendent.

Declaration: `procedure SaveToIni(Ini: TCustomIniFile; Section: string);  Virtual`

Visibility: public

Description: `SaveToIni` does the actual writing. It writes the number of elements in the global section (as specified by the `Section` argument), as well as the section name for each item in the list. The item names are written using the Prefix (1268) property for the key. After this it calls the SaveToIni (1270) method of all TIniCollectionItem (1270) instances.

This means that the global section of the .ini file will look something like this:

```
[globalsection]
Count=3
Prefix1=SectionPrefixFirstItemName
Prefix2=SectionPrefixSecondItemName
Prefix3=SectionPrefixThirdItemName
```

This construct allows to re-use an ini file for multiple collections.

After this method is called, the GlobalSection (1269) property contains the value of `Section`, it will be used in the Save (1267) method.

See also: TIniCollectionItem.SaveToIni (1270), TIniCollection.Save (1267)

40.5.7 TIniCollection.SaveToFile

Synopsis: Save collection to a file in .ini file format.

Declaration: `procedure SaveToFile(const AFileName: string; const Section: string)`

Visibility: public

Description: `SaveToFile` will create a `TMemIniFile` instance with the `AFileName` argument as a filename. This instance is passed on to the SaveToIni (1267) method, together with the `Section` argument, to do the actual saving.

Errors: An exception may be raised if the path in `AFileName` does not exist.

See also: TIniCollection.SaveToIni (1267), TIniCollection.LoadFromFile (1268)

40.5.8 TIniCollection.LoadFromIni

Synopsis: Load collection from a file in .ini file format.

Declaration: `procedure LoadFromIni(Ini: TCustomIniFile; Section: string);  Virtual`

Visibility: public

Description: `LoadFromIni` will load the collection from the `Ini` instance. It first clears the collection, and reads the number of items from the global section with the name as passed through the `Section` argument. After this, an item is created and added to the collection, and its data is read by calling the TIniCollectionItem.LoadFromIni (1270) method, passing the appropriate section name as found in the global section.

The description of the global section can be found in the TIniCollection.SaveToIni (1267) method description.

See also: TIniCollection.LoadFromFile (1268), TIniCollectionItem.LoadFromIni (1270), TIniCollection.SaveToIni (1267)

40.5.9 TIniCollection.LoadFromFile

Synopsis: Load collection from file.

Declaration: `procedure LoadFromFile(const AFileName: string; const Section: string)`

Visibility: public

Description: `LoadFromFile` creates a `TMemIniFile` instance using `AFileName` as the filename. It calls LoadFromIni (1268) using this instance and `Section` as the parameters.

See also: TIniCollection.LoadFromIni (1268), TIniCollection.Load (1266), TIniCollection.SaveToIni (1267), TIniCollection.SaveToFile (1267)

40.5.10 TIniCollection.Prefix

Synopsis: Prefix used in global section.

Declaration: `Property Prefix : string`

Visibility: public

Access: Read

Description: `Prefix` is used when writing the section names of the items in the collection to the global section, or when reading the names from the global section. If the prefix is set to `Item` then the global section might look something like this:

```
[MyCollection]
Count=2
Item1=FirstItem
Item2=SecondItem
```

A descendent of `TIniCollection` should set the value of this property, it cannot be empty.

See also: TIniCollection.SectionPrefix (1269), TIniCollection.GlobalSection (1269)

40.5.11 TIniCollection.SectionPrefix

Synopsis: Prefix string for section names.

Declaration: `Property SectionPrefix : string`

Visibility: public

Access: Read

Description: `SectionPrefix` is a string that is prepended to the section name specified using the TIniCollectionItem.SectionName (1271) property. The two elements form the actual section name where the collection items are stored. The value can be an empty string (**''**) if a Prefix is not needed in the realized SectionName.

See also: TIniCollection.GlobalSection (1269), TIniCollectionItem.SectionName (1271)

40.5.12 TIniCollection.FileName

Synopsis: Filename of the collection.

Declaration: `Property FileName : string`

Visibility: public

Access: Read,Write

Description: `FileName` is the filename as used in the last LoadFromFile (1268) or SaveToFile (1267) operation. It is used in the Load (1266) or Save (1267) calls.

See also: Save (1267), LoadFromFile (1268), SaveToFile (1267), Load (1266)

40.5.13 TIniCollection.GlobalSection

Synopsis: Name of the global section.

Declaration: `Property GlobalSection : string`

Visibility: public

Access: Read,Write

Description: `GlobalSection` contains the value of the `Section` argument in the LoadFromIni (1268) or SaveToIni (1267) calls. It's used in the Load (1266) or Save (1267) calls.

See also: Save (1267), LoadFromFile (1268), SaveToFile (1267), Load (1266)

40.6 TIniCollectionItem

40.6.1 Description

`TIniCollectionItem` is a #rtl.classes.tcollectionitem (**??**) descendent which has some extra methods for saving/loading the item to or from an .ini file.

To use this class, a descendent should be made, and the SaveToIni (1270) and LoadFromIni (1270) methods should be overridden. They should implement the actual loading and saving. The loading and saving is always initiated by the methods in TIniCollection (1266), TIniCollection.LoadFromIni (1268) and TIniCollection.SaveToIni (1267) respectively.

See also: TIniCollection (1266), TIniCollectionItem.SaveToIni (1270), TIniCollectionItem.LoadFromIni (1270), TIniCollection.LoadFromIni (1268), TIniCollection.SaveToIni (1267)

40.6.2 Method overview

Page	Method	Description
1271	LoadFromFile	Load item from a file.
1270	LoadFromIni	Method called when the item must be loaded.
1271	SaveToFile	Save item to a file.
1270	SaveToIni	Method called when the item must be saved.

40.6.3 Property overview

Page	Properties	Access	Description
1271	SectionName	rw	Default section name.

40.6.4 TIniCollectionItem.SaveToIni

Synopsis: Method called when the item must be saved.

Declaration:
```
procedure SaveToIni(Ini: TCustomIniFile; Section: string);  Virtual
                        ;  Abstract
```

Visibility: public

Description: `SaveToIni` is called by TIniCollection.SaveToIni (1267) when it saves this item. Descendent classes should override this method to save the data they need to save. All write methods of the `TCustomIniFile` instance passed in `Ini` can be used, as long as the writing happens in the section passed in `Section`.

Errors: No checking is done to see whether the values are actually written to the correct section.

See also: TIniCollection.SaveToIni (1267), LoadFromIni (1270), SaveToFile (1271), LoadFromFile (1271)

40.6.5 TIniCollectionItem.LoadFromIni

Synopsis: Method called when the item must be loaded.

Declaration:
```
procedure LoadFromIni(Ini: TCustomIniFile; Section: string);  Virtual
                          ;  Abstract
```

Visibility: public

Description: `LoadFromIni` is called by TIniCollection.LoadFromIni (1268) when it saves this item. Descendent classes should override this method to load the data they need to load. All read methods of the `TCustomIniFile` instance passed in `Ini` can be used, as long as the reading happens in the section passed in `Section`.

Errors: No checking is done to see whether the values are actually read from the correct section.

See also: TIniCollection.LoadFromIni (1268), SaveToIni (1270), LoadFromFile (1271), SaveToFile (1271)

40.6.6 TIniCollectionItem.SaveToFile

Synopsis: Save item to a file.

Declaration: `procedure SaveToFile(const FileName: string; const Section: string)`

Visibility: public

Description: `SaveToFile` creates an instance of `TIniFile` with the indicated `FileName` calls SaveToIni (1270) to save the item to the indicated file in .ini format under the section `Section`

Errors: An exception can occur if the file is not writeable.

See also: SaveToIni (1270), LoadFromFile (1271)

40.6.7 TIniCollectionItem.LoadFromFile

Synopsis: Load item from a file.

Declaration: `procedure LoadFromFile(const FileName: string; const Section: string)`

Visibility: public

Description: `LoadFromFile` creates an instance of `TMemIniFile` and calls LoadFromIni (1270) to load the item from the indicated file in .ini format from the section `Section`.

Errors: None.

See also: SaveToFile (1271), LoadFromIni (1270)

40.6.8 TIniCollectionItem.SectionName

Synopsis: Default section name.

Declaration: `Property SectionName : string`

Visibility: public

Access: Read,Write

Description: `SectionName` is the section name under which the item will be saved or from which it should be read. The read/write functions should be overridden in descendents to determine a unique section name within the .ini file.

See also: SaveToFile (1271), LoadFromIni (1270)

40.7 TNamedIniCollection

40.7.1 Description

`TNamedIniCollection` is the collection to go with the TNamedIniCollectionItem (1273) item class. it provides some functions to look for items based on the UserData (1272) or based on the Name (1272).

See also: TNamedIniCollectionItem (1273), IndexOfUserData (1272), IndexOfName (1272)

40.7.2 Method overview

Page	Method	Description
1273	FindByName	Return the item based on its name.
1273	FindByUserData	Return the item based on its `UserData`.
1272	IndexOfName	Search for an item, based on its name, and return its position.
1272	IndexOfUserData	Search for an item based on it's `UserData` property.

40.7.3 Property overview

Page	Properties	Access	Description
1273	NamedItems	rw	Indexed access to the `TNamedIniCollectionItem` items.

40.7.4 TNamedIniCollection.IndexOfUserData

Synopsis: Search for an item based on it's `UserData` property.

Declaration: `function IndexOfUserData(UserData: TObject) : Integer`

Visibility: public

Description: `IndexOfUserData` searches the list of items and returns the index of the item which has `UserData` in its UserData (1272) property. If no such item exists, -1 is returned.

Note that the (linear) search starts at the last element and works it's way back to the first.

Errors: If no item exists, -1 is returned.

See also: IndexOfName (1272), TNamedIniCollectionItem.UserData (1274)

40.7.5 TNamedIniCollection.IndexOfName

Synopsis: Search for an item, based on its name, and return its position.

Declaration: `function IndexOfName(const AName: string) : Integer`

Visibility: public

Description: `IndexOfName` searches the list of items and returns the index of the item which has name equal to `AName` (case insensitive). If no such item exists, -1 is returned.

Note that the (linear) search starts at the last element and works it's way back to the first.

Errors: If no item exists, -1 is returned.

See also: IndexOfUserData (1272), TNamedIniCollectionItem.Name (1274)

40.7.6 TNamedIniCollection.FindByName

Synopsis: Return the item based on its name.

Declaration: `function FindByName(const AName: string) : TNamedIniCollectionItem`

Visibility: public

Description: `FindByName` returns the collection item whose name matches `AName` (case insensitive match). It calls IndexOfName (1272) and returns the item at the found position. If no item is found, `Nil` is returned.

Errors: If no item is found, `Nil` is returned.

See also: IndexOfName (1272), FindByUserData (1273)

40.7.7 TNamedIniCollection.FindByUserData

Synopsis: Return the item based on its `UserData`.

Declaration: `function FindByUserData(UserData: TObject) : TNamedIniCollectionItem`

Visibility: public

Description: `FindByName` returns the collection item whose UserData (1274) property value matches the `UserData` parameter. If no item is found, `Nil` is returned.

Errors: If no item is found, `Nil` is returned.

40.7.8 TNamedIniCollection.NamedItems

Synopsis: Indexed access to the `TNamedIniCollectionItem` items.

Declaration: `Property NamedItems[Index: Integer]: TNamedIniCollectionItem; default`

Visibility: public

Access: Read,Write

Description: `NamedItem` is the default property of the `TNamedIniCollection` collection. It allows indexed access to the TNamedIniCollectionItem (1273) items. The index is zero based.

See also: TNamedIniCollectionItem (1273)

40.8 TNamedIniCollectionItem

40.8.1 Description

`TNamedIniCollectionItem` is a TIniCollectionItem (1270) descent with a published name property. The name is used as the section name when saving the item to the ini file.

See also: TIniCollectionItem (1270)

40.8.2 Property overview

Page	Properties	Access	Description
1274	Name	rw	Name of the item.
1274	UserData	rw	User-defined data.

40.8.3 TNamedIniCollectionItem.UserData

Synopsis: User-defined data.

Declaration: `Property UserData : TObject`

Visibility: public

Access: Read,Write

Description: `UserData` can be used to associate an arbitrary object with the item - much like the `Objects` property of a `TStrings`.

40.8.4 TNamedIniCollectionItem.Name

Synopsis: Name of the item.

Declaration: `Property Name : string`

Visibility: published

Access: Read,Write

Description: `Name` is the name of this item. It is also used as the section name when writing the collection item to the .ini file.

See also: TNamedIniCollectionItem.UserData (1274)

Chapter 41

Reference for unit 'IniFiles'

41.1 Used units

Table 41.1: Used units by unit 'IniFiles'

Name	Page
Classes	??
Contnrs	284
System	??
sysutils	??

41.2 Overview

IniFiles provides support for handling .ini files. It contains an implementation completely independent of the Windows API for handling such files. The basic (abstract) functionality is defined in TCustomInifile (1277) and is implemented in TIniFile (1290) and TMemIniFile (1299). The API presented by these components is Delphi compatible.

41.3 Constants, types and variables

41.3.1 Constants

```
ifoWriteStringBoolean = ifoStringBoolean
```

41.3.2 Types

```
TIniFileOption = (ifoStripComments,ifoStripInvalid,ifoEscapeLineFeeds
  ,
                  ifoCaseSensitive,ifoStripQuotes,
  ifoFormatSettingsActive,ifoStringBoolean)
```

Table 41.2: Enumeration values for type TIniFileOption

Value	Explanation
ifoCaseSensitive	Key and section names are case sensitive.
ifoEscapeLineFeeds	Observe backslash as linefeed escape character.
ifoFormatSettingsActive	Observe the values in FormatSettings.
ifoStringBoolean	
ifoStripComments	Strip comments from file.
ifoStripInvalid	Strip invalid lines from file.
ifoStripQuotes	Strip double quotes from values.

`TIniFileOption` enumerates the possible options when creating a new TCustomIniFile (1277) instance.

ifoStripComments Strip comments from file.

ifoStripInvalid Strip invalid lines from file.

ifoEscapeLineFeeds Observe backslash as linefeed escape character.

ifoCaseSensitive Key and section names are case sensitive.

ifoStripQuotes Strip double quotes from values.

ifoFormatSettingsActive Observe the values in FormatSettings.

```
TIniFileOptions = Set of TIniFileOption
```

`TIniFIleOptions` is the set for TIniFileOption (1275). It is used in the TCUstomIniFile.Create (1278) constructor and TCustomIniFile.Options (1287) property.

```
TSectionValuesOption = (svoIncludeComments,svoIncludeInvalid,
  svoIncludeQuotes)
```

Table 41.3: Enumeration values for type TSectionValuesOption

Value	Explanation
svoIncludeComments	Include comment lines.
svoIncludeInvalid	Include invalid lines.
svoIncludeQuotes	Include existing quotes around values.

`TSectionValuesOption` is used to control the behaviour of TCustomIniFile.ReadSectionValues (1285)

svoIncludeComments Include comment lines.

svoIncludeInvalid Include invalid lines.

svoIncludeQuotes Include existing quotes around values.

```
TSectionValuesOptions = Set of TSectionValuesOption
```

`TSectionValuesOptions` is the set for TSectionValuesOptions (1276). It is used in the TCUstomIniFile.ReadSectionValues (1285) call.

41.4 TCustomIniFile

41.4.1 Description

`TCustomIniFile` implements all calls for manipulating a .ini. It does not implement any of this behaviour, the behaviour must be implemented in a descendent class like TIniFile (1290) or TMemIniFile (1299).

Since `TCustomIniFile` is an abstract class, it should never be created directly. Instead, one of the `TIniFile` or `TMemIniFile` classes should be created.

See also: TIniFile (1290), TMemIniFile (1299)

41.4.2 Method overview

Page	Method	Description
1278	Create	Instantiate a new instance of `TCustomIniFile`.
1286	DeleteKey	Delete a key from a section.
1279	Destroy	Remove the `TCustomIniFile` instance from memory.
1285	EraseSection	Clear a section.
1283	ReadBinaryStream	Read binary data.
1281	ReadBool	
1282	ReadDate	Read a date value.
1282	ReadDateTime	Read a Date/Time value.
1282	ReadFloat	Read a floating point value.
1280	ReadInt64	Read an Int64 value.
1280	ReadInteger	Read an integer value from the file.
1284	ReadSection	Read the key names in a section.
1285	ReadSections	Read the list of sections.
1285	ReadSectionValues	Read names and values of a section.
1279	ReadString	Read a string valued key.
1282	ReadTime	Read a time value.
1279	SectionExists	Check if a section exists.
1279	SetBoolStringValues	Set the boolean string values to use when writing to file.
1286	UpdateFile	Update the file on disk.
1286	ValueExists	Check if a value exists.
1284	WriteBinaryStream	Write binary data.
1281	WriteBool	Write boolean value.
1283	WriteDate	Write date value.
1283	WriteDateTime	Write date/time value.
1284	WriteFloat	Write a floating-point value.
1281	WriteInt64	Write an Int64 value.
1280	WriteInteger	Write an integer value.
1280	WriteString	Write a string value.
1284	WriteTime	Write time value.

41.4.3 Property overview

Page	Properties	Access	Description
1289	BoolFalseStrings	rw	Strings to recognize as boolean `False` values.
1288	BoolTrueStrings	rw	Strings to recognize as boolean `True` values.
1287	CaseSensitive	rw	Are key and section names case sensitive.
1286	Encoding	rw	Encoding of the ini file.
1287	EscapeLineFeeds	r	Should linefeeds be escaped ?
1287	FileName	r	Name of the .ini file.
1288	FormatSettingsActive	rw	Is `FormatSettings` used or not.
1287	Options	rw	Options currently in effect.
1289	OwnsEncoding	r	Does the ini file instance own the encoding ?
1288	StripQuotes	rw	Should quotes be stripped from string values.

41.4.4 TCustomIniFile.Create

Synopsis: Instantiate a new instance of `TCustomIniFile`.

Declaration:
```
constructor Create(const AFileName: string;
                   ADefaultEncoding: TEncoding;
                   AOptions: TIniFileOptions)
constructor Create(const AFileName: string;
                   ADefaultEncoding: TEncoding; AOwnsEncoding: Boolean;
                   AOptions: TIniFileOptions)
constructor Create(const AFileName: string; AOptions: TIniFileOptions)
                   ;  Virtual
constructor Create(const AFileName: string; AEscapeLineFeeds: Boolean)
                   ;  Virtual
```

Visibility: public

Description: `Create` creates a new instance of `TCustomIniFile` and loads it with the data from `AFileName`, if this file exists. If the `ifoEscapeLineFeeds` option is present in `AOptions` or `AEscapeLineFeeds` parameter is `True`, then lines which have their end-of-line markers escaped with a backslash, will be concatenated. This means that the following 2 lines

```
Description=This is a \
line with a long text
```

is equivalent to

```
Description=This is a line with a long text
```

By default, not escaping of linefeeds is performed (for Delphi compatibility)

Default options for the TCustomIniFile.Options (1287) property can be specified in `AOptions`.

A Default string encoding can be specified in `aEncoding`. If `aOwnsEncoding` is `True` the encoding will be freed when the ini file instance is destroyed.

Errors: If the file cannot be read, an exception may be raised.

See also: Destroy (1279), TCustomIniFile.Options (1287)

41.4.5 TCustomIniFile.Destroy

Synopsis: Remove the `TCustomIniFile` instance from memory.

Declaration:
```
destructor Destroy;  Override
```

Visibility: public

Description: `Destroy` cleans up all internal structures and then calls the inherited `Destroy`.

See also: TCustomIniFile (1277)

41.4.6 TCustomIniFile.SetBoolStringValues

Synopsis: Set the boolean string values to use when writing to file.

Declaration:
```
procedure SetBoolStringValues(ABoolValue: Boolean;
                              Values: Array of string)
```

Visibility: public

Description: `SetBoolStringValues` Sets the strings to recognize as boolean variable `ABoolValue` to `Values`. When reading boolean values, these values are reported as `ABoolValue` in TCustomIniFile.ReadBool (1281). The first value in the list is used to write the boolean `ABoolValue` value.

To ensure correct operation, this function should be called with values for both `True` and `False`.

The string values are ignored if `ifoWriteStringBoolean` is not in the ini file options (1287).

See also: TCustomIniFile.ReadBool (1281), TCustomIniFile.WriteBool (1281), TCustomIniFile.BoolTrueStrings (1288), TCustomIniFile.BoolFalseStrings (1289), TCustomIniFile.Options (1287)

41.4.7 TCustomIniFile.SectionExists

Synopsis: Check if a section exists.

Declaration:
```
function SectionExists(const Section: string) : Boolean;  Virtual
```

Visibility: public

Description: `SectionExists` returns `True` if a section with name `Section` exists, and contains keys. (comments are not considered keys)

See also: TCustomIniFile.ValueExists (1286)

41.4.8 TCustomIniFile.ReadString

Synopsis: Read a string valued key.

Declaration:
```
function ReadString(const Section: string; const Ident: string;
                    const Default: string) : string;  Virtual;  Abstract
```

Visibility: public

Description: `ReadString` reads the key `Ident` in section `Section`, and returns the value as a string. If the specified key or section do not exist, then the value in `Default` is returned. Note that if the key exists, but is empty, an empty string will be returned.

See also: WriteString (1280), ReadInteger (1280), ReadBool (1281), ReadDate (1282), ReadDateTime (1282), ReadTime (1282), ReadFloat (1282), ReadBinaryStream (1283)

41.4.9 TCustomIniFile.WriteString

Synopsis: Write a string value.

Declaration:
```
procedure WriteString(const Section: string; const Ident: string;
                      const Value: string);  Virtual;  Abstract
```

Visibility: public

Description: `WriteString` writes the string `Value` with the name `Ident` to the section `Section`, overwriting any previous value that may exist there. The section will be created if it does not exist.

Note that it is not possible to write strings with newline characters in them. Newlines can be read from a .ini file, but there is no support for writing them.

See also: ReadString (1279), WriteInteger (1280), WriteBool (1281), WriteDate (1283), WriteDateTime (1283), WriteTime (1284), WriteFloat (1284), WriteBinaryStream (1284)

41.4.10 TCustomIniFile.ReadInteger

Synopsis: Read an integer value from the file.

Declaration:
```
function ReadInteger(const Section: string; const Ident: string;
                     Default: LongInt) : LongInt;  Virtual
```

Visibility: public

Description: `ReadInteger` reads the key `Ident` in section `Section`, and returns the value as an integer. If the specified key or section do not exist, then the value in `Default` is returned. If the key exists, but contains an invalid integer value, `Default` is also returned.

See also: WriteInteger (1280), ReadString (1279), ReadBool (1281), ReadDate (1282), ReadDateTime (1282), ReadTime (1282), ReadFloat (1282), ReadBinaryStream (1283)

41.4.11 TCustomIniFile.WriteInteger

Synopsis: Write an integer value.

Declaration:
```
procedure WriteInteger(const Section: string; const Ident: string;
                       Value: LongInt);  Virtual
```

Visibility: public

Description: `WriteInteger` writes the integer `Value` with the name `Ident` to the section `Section`, overwriting any previous value that may exist there. The section will be created if it does not exist.

See also: ReadInteger (1280), WriteString (1280), WriteBool (1281), WriteDate (1283), WriteDateTime (1283), WriteTime (1284), WriteFloat (1284), WriteBinaryStream (1284)

41.4.12 TCustomIniFile.ReadInt64

Synopsis: Read an Int64 value.

Declaration:
```
function ReadInt64(const Section: string; const Ident: string;
                   Default: Int64) : Int64;  Virtual
```

Visibility: public

Description: `ReadInt64` reads a signed 64-bit integer value from the ini file. The value is searched in the `Section` section, with key `Ident`.

If the value is not found at the specified `Section`, `Ident` pair, or the value is not a `Int64` value then the `Default` value is returned instead.

This function is needed because ReadInteger (1280) reads at most a 32-bit value.

See also: TCustomIniFile.ReadInteger (1280), TCustomIniFile.WriteInt64 (1281)

41.4.13 TCustomIniFile.WriteInt64

Synopsis: Write an Int64 value.

Declaration:
```
procedure WriteInt64(const Section: string; const Ident: string;
                     Value: Int64);  Virtual
```

Visibility: public

Description: `WriteInt64` writes `Value` as a signed 64-bit integer value to section `Section`, key `Ident`.

See also: TCustomIniFile.WriteInteger (1280), TCustomIniFile.ReadInt64 (1280)

41.4.14 TCustomIniFile.ReadBool

Synopsis:

Declaration:
```
function ReadBool(const Section: string; const Ident: string;
                  Default: Boolean) : Boolean;  Virtual
```

Visibility: public

Description: `ReadString` reads the key `Ident` in section `Section`, and returns the value as a boolean (valid values are 0 and 1). If the specified key or section do not exist, then the value in `Default` is returned. If the key exists, but contains an invalid integer value, `False` is also returned.

See also: WriteBool (1281), ReadInteger (1280), ReadString (1279), ReadDate (1282), ReadDateTime (1282), ReadTime (1282), ReadFloat (1282), ReadBinaryStream (1283)

41.4.15 TCustomIniFile.WriteBool

Synopsis: Write boolean value.

Declaration:
```
procedure WriteBool(const Section: string; const Ident: string;
                    Value: Boolean);  Virtual
```

Visibility: public

Description: `WriteBool` writes the boolean `Value` with the name `Ident` to the section `Section`, overwriting any previous value that may exist there. The section will be created if it does not exist.

See also: ReadBool (1281), WriteInteger (1280), WriteString (1280), WriteDate (1283), WriteDateTime (1283), WriteTime (1284), WriteFloat (1284), WriteBinaryStream (1284)

41.4.16 TCustomIniFile.ReadDate

Synopsis: Read a date value.

Declaration:
```
function ReadDate(const Section: string; const Ident: string;
                  Default: TDateTime) : TDateTime;  Virtual
```

Visibility: public

Description: `ReadDate` reads the key `Ident` in section `Section`, and returns the value as a date (`TDateTime`). If the specified key or section do not exist, then the value in `Default` is returned. If the key exists, but contains an invalid date value, `Default` is also returned. The international settings of the SysUtils are taken into account when deciding if the read value is a correct date.

See also: WriteDate (1283), ReadInteger (1280), ReadBool (1281), ReadString (1279), ReadDateTime (1282), ReadTime (1282), ReadFloat (1282), ReadBinaryStream (1283)

41.4.17 TCustomIniFile.ReadDateTime

Synopsis: Read a Date/Time value.

Declaration:
```
function ReadDateTime(const Section: string; const Ident: string;
                      Default: TDateTime) : TDateTime;  Virtual
```

Visibility: public

Description: `ReadDateTime` reads the key `Ident` in section `Section`, and returns the value as a date/time (`TDateTime`). If the specified key or section do not exist, then the value in `Default` is returned. If the key exists, but contains an invalid date/time value, `Default` is also returned. The international settings of the SysUtils are taken into account when deciding if the read value is a correct date/time.

See also: WriteDateTime (1283), ReadInteger (1280), ReadBool (1281), ReadDate (1282), ReadString (1279), ReadTime (1282), ReadFloat (1282), ReadBinaryStream (1283)

41.4.18 TCustomIniFile.ReadFloat

Synopsis: Read a floating point value.

Declaration:
```
function ReadFloat(const Section: string; const Ident: string;
                   Default: Double) : Double;  Virtual
```

Visibility: public

Description: `ReadFloat` reads the key `Ident` in section `Section`, and returns the value as a float (`Double`). If the specified key or section do not exist, then the value in `Default` is returned. If the key exists, but contains an invalid float value, `Default` is also returned. The international settings of the SysUtils are taken into account when deciding if the read value is a correct float.

See also: WriteFloat (1284), ReadInteger (1280), ReadBool (1281), ReadDate (1282), ReadDateTime (1282), ReadTime (1282), ReadString (1279), ReadBinaryStream (1283)

41.4.19 TCustomIniFile.ReadTime

Synopsis: Read a time value.

Declaration:
```
function ReadTime(const Section: string; const Ident: string;
                  Default: TDateTime) : TDateTime;  Virtual
```

Visibility: public

Description: `ReadTime` reads the key `Ident` in section `Section`, and returns the value as a time (`TDateTime`). If the specified key or section do not exist, then the value in `Default` is returned. If the key exists, but contains an invalid time value, `Default` is also returned. The international settings of the SysUtils are taken into account when deciding if the read value is a correct time.

See also: WriteTime (1284), ReadInteger (1280), ReadBool (1281), ReadDate (1282), ReadDateTime (1282), ReadString (1279), ReadFloat (1282), ReadBinaryStream (1283)

41.4.20 TCustomIniFile.ReadBinaryStream

Synopsis: Read binary data.

Declaration:
```
function ReadBinaryStream(const Section: string; const Name: string;
                          Value: TStream) : Integer;  Virtual
```

Visibility: public

Description: `ReadBinaryStream` reads the key `Name` in section `Section`, and returns the value in the stream `Value`. If the specified key or section do not exist, then the contents of `Value` are left untouched. The stream is not cleared prior to adding data to it.

The data is interpreted as a series of 2-byte hexadecimal values, each representing a byte in the data stream, i.e, it should always be an even number of hexadecimal characters.

See also: WriteBinaryStream (1284), ReadInteger (1280), ReadBool (1281), ReadDate (1282), ReadDateTime (1282), ReadTime (1282), ReadFloat (1282), ReadString (1279)

41.4.21 TCustomIniFile.WriteDate

Synopsis: Write date value.

Declaration:
```
procedure WriteDate(const Section: string; const Ident: string;
                    Value: TDateTime);  Virtual
```

Visibility: public

Description: `WriteDate` writes the date `Value` with the name `Ident` to the section `Section`, overwriting any previous value that may exist there. The section will be created if it does not exist. The date is written using the internationalization settings in the SysUtils unit.

See also: ReadDate (1282), WriteInteger (1280), WriteBool (1281), WriteString (1280), WriteDateTime (1283), WriteTime (1284), WriteFloat (1284), WriteBinaryStream (1284)

41.4.22 TCustomIniFile.WriteDateTime

Synopsis: Write date/time value.

Declaration:
```
procedure WriteDateTime(const Section: string; const Ident: string;
                        Value: TDateTime);  Virtual
```

Visibility: public

Description: `WriteDateTime` writes the date/time `Value` with the name `Ident` to the section `Section`, overwriting any previous value that may exist there. The section will be created if it does not exist. The date/time is written using the internationalization settings in the SysUtils unit.

See also: ReadDateTime (1282), WriteInteger (1280), WriteBool (1281), WriteDate (1283), WriteString (1280), WriteTime (1284), WriteFloat (1284), WriteBinaryStream (1284)

41.4.23 TCustomIniFile.WriteFloat

Synopsis: Write a floating-point value.

Declaration:
```
procedure WriteFloat(const Section: string; const Ident: string;
                     Value: Double);  Virtual
```

Visibility: public

Description: `WriteFloat` writes the time `Value` with the name `Ident` to the section `Section`, overwriting any previous value that may exist there. The section will be created if it does not exist. The floating point value is written using the internationalization settings in the SysUtils unit.

See also: ReadFloat (1282), WriteInteger (1280), WriteBool (1281), WriteDate (1283), WriteDateTime (1283), WriteTime (1284), WriteString (1280), WriteBinaryStream (1284)

41.4.24 TCustomIniFile.WriteTime

Synopsis: Write time value.

Declaration:
```
procedure WriteTime(const Section: string; const Ident: string;
                    Value: TDateTime);  Virtual
```

Visibility: public

Description: `WriteTime` writes the time `Value` with the name `Ident` to the section `Section`, overwriting any previous value that may exist there. The section will be created if it does not exist. The time is written using the internationalization settings in the SysUtils unit.

See also: ReadTime (1282), WriteInteger (1280), WriteBool (1281), WriteDate (1283), WriteDateTime (1283), WriteString (1280), WriteFloat (1284), WriteBinaryStream (1284)

41.4.25 TCustomIniFile.WriteBinaryStream

Synopsis: Write binary data.

Declaration:
```
procedure WriteBinaryStream(const Section: string; const Name: string;
                            Value: TStream);  Virtual
```

Visibility: public

Description: `WriteBinaryStream` writes the binary data in `Value` with the name `Ident` to the section `Section`, overwriting any previous value that may exist there. The section will be created if it does not exist.

The binary data is encoded using a 2-byte hexadecimal value per byte in the data stream. The data stream must be seekable, so it's size can be determined. The data stream is not repositioned, it must be at the correct position.

See also: ReadBinaryStream (1283), WriteInteger (1280), WriteBool (1281), WriteDate (1283), WriteDateTime (1283), WriteTime (1284), WriteFloat (1284), WriteString (1280)

41.4.26 TCustomIniFile.ReadSection

Synopsis: Read the key names in a section.

Declaration:
```
procedure ReadSection(const Section: string; Strings: TStrings)
                      ;  Virtual;  Abstract
```

Visibility: public

Description: `ReadSection` will return the names of the keys in section `Section` in `Strings`, one string per key. If a non-existing section is specified, the list is cleared. To return the values of the keys as well, the ReadSectionValues (1285) method should be used.

See also: ReadSections (1285), SectionExists (1279), ReadSectionValues (1285)

41.4.27 TCustomIniFile.ReadSections

Synopsis: Read the list of sections.

Declaration:
```
procedure ReadSections(Strings: TStrings);  Virtual;  Abstract
```

Visibility: public

Description: `ReadSections` returns the names of existing sections in `Strings`. It also returns names of empty sections.

See also: SectionExists (1279), ReadSectionValues (1285), ReadSection (1284)

41.4.28 TCustomIniFile.ReadSectionValues

Synopsis: Read names and values of a section.

Declaration:
```
procedure ReadSectionValues(const Section: string; Strings: TStrings;
                            Options: TSectionValuesOptions);  Virtual
                            ;  Overload
procedure ReadSectionValues(const Section: string; Strings: TStrings)
                            ;  Virtual;  Overload
```

Visibility: public

Description: `ReadSectionValues` returns the keys and their values in the section `Section` in `Strings`. They are returned as `Key=Value` strings, one per key, so the `Values` property of the stringlist can be used to read the values. To retrieve just the names of the available keys, ReadSection (1284) can be used.

See also: SectionExists (1279), ReadSections (1285), ReadSection (1284)

41.4.29 TCustomIniFile.EraseSection

Synopsis: Clear a section.

Declaration:
```
procedure EraseSection(const Section: string);  Virtual;  Abstract
```

Visibility: public

Description: `EraseSection` deletes all values from the section named `Section` and removes the section from the ini file. If the section didn't exist prior to a call to `EraseSection`, nothing happens.

See also: SectionExists (1279), ReadSections (1285), DeleteKey (1286)

41.4.30 TCustomIniFile.DeleteKey

Synopsis: Delete a key from a section.

Declaration:
```
procedure DeleteKey(const Section: string; const Ident: string)
                          ;  Virtual;  Abstract
```

Visibility: public

Description: `DeleteKey` deletes the key `Ident` from section `Section`. If the key or section didn't exist prior to the `DeleteKey` call, nothing happens.

See also: EraseSection (1285)

41.4.31 TCustomIniFile.UpdateFile

Synopsis: Update the file on disk.

Declaration:
```
procedure UpdateFile;  Virtual;  Abstract
```

Visibility: public

Description: `UpdateFile` writes the in-memory image of the ini-file to disk. To speed up operation of the inifile class, the whole ini-file is read into memory when the class is created, and all operations are performed in-memory. If `CacheUpdates` is set to `True`, any changes to the inifile are only in memory, until they are committed to disk with a call to `UpdateFile`. If `CacheUpdates` is set to `False`, then all operations which cause a change in the .ini file will immediately be committed to disk with a call to `UpdateFile`. Since the whole file is written to disk, this may have serious impact on performance.

See also: CacheUpdates (1294)

41.4.32 TCustomIniFile.ValueExists

Synopsis: Check if a value exists.

Declaration:
```
function ValueExists(const Section: string; const Ident: string)
                           : Boolean;  Virtual
```

Visibility: public

Description: `ValueExists` checks whether the key `Ident` exists in section `Section`. It returns `True` if a key was found, or `False` if not. The key may be empty.

See also: SectionExists (1279)

41.4.33 TCustomIniFile.Encoding

Synopsis: Encoding of the ini file.

Declaration:
```
Property Encoding : TEncoding
```

Visibility: public

Access: Read,Write

Description: `Encoding` is the encoding specified in the constructor. It cannot be changed during the lifetime of the instance.

See also: TCustomIniFile.Create (1278), TCustomIniFile.OwnsEncoding (1289)

41.4.34 TCustomIniFile.FileName

Synopsis: Name of the .ini file.

Declaration: `Property FileName : string`

Visibility: public

Access: Read

Description: `FileName` is the name of the ini file on disk. It should be specified when the `TCustomIniFile` instance is created. Contrary to the Delphi implementation, if no path component is present in the filename, the filename is not searched in the windows directory.

See also: Create (1278)

41.4.35 TCustomIniFile.Options

Synopsis: Options currently in effect.

Declaration: `Property Options : TIniFileOptions`

Visibility: public

Access: Read,Write

Description: `Options` is the set of options currently in effect. See TInifileOption (1275) for a list of allowed options. The initial value of this property can be specified using the constructor of the class, TCustomIniFile.Create (1278). Not all options can be specified after the ini file object was created.

See also: TInifileOption (1275), TInifileOptions (1276), TCustomIniFile.Create (1278)

41.4.36 TCustomIniFile.EscapeLineFeeds

Synopsis: Should linefeeds be escaped ?

Declaration: `Property EscapeLineFeeds : Boolean; deprecated;`

Visibility: public

Access: Read

Description: `EscapeLineFeeds` determines whether escaping of linefeeds is enabled: For a description of this feature, see Create (1278), as the value of this property must be specified when the `TCustomIniFile` instance is created.

By default, `EscapeLineFeeds` is `False`.

See also: Create (1278), CaseSensitive (1287)

41.4.37 TCustomIniFile.CaseSensitive

Synopsis: Are key and section names case sensitive.

Declaration: `Property CaseSensitive : Boolean; deprecated;`

Visibility: public

Access: Read,Write

Description: `CaseSensitive` determines whether searches for sections and keys are performed case-sensitive or not. By default, they are not case sensitive.

See also: EscapeLineFeeds (1287)

41.4.38 TCustomIniFile.StripQuotes

Synopsis: Should quotes be stripped from string values.

Declaration: `Property StripQuotes : Boolean; deprecated;`

Visibility: public

Access: Read,Write

Description: `StripQuotes` determines whether quotes around string values are stripped from the value when reading the values from file. By default, quotes are not stripped (this is Delphi and Windows compatible).

41.4.39 TCustomIniFile.FormatSettingsActive

Synopsis: Is `FormatSettings` used or not.

Declaration: `Property FormatSettingsActive : Boolean; deprecated;`

Visibility: public

Access: Read,Write

Description: `FormatSettingsActive` can be set to `True` to use the TCustomIniFile.FormatSettings (**??**) field when reading and/or writing values of type date/time or float. If the setting is set to `False` then the defaults specified in the sysutils unit are used.

41.4.40 TCustomIniFile.BoolTrueStrings

Synopsis: Strings to recognize as boolean `True` values.

Declaration: `Property BoolTrueStrings : TStringArray`

Visibility: public

Access: Read,Write

Description: `BoolTrueStrings` is a list of strings that will be recognized as boolean `True` value in TCustomIniFile.ReadBool (1281) The first string in the list will be used when writing a `True` boolean value in TCustomIniFile.WriteBool (1281).

The string values are ignored if `ifoWriteStringBoolean` is not in the ini file options (1287).

See also: TCustomIniFile.SetBoolStringValues (1279), TCustomIniFile.BoolFalseStrings (1289), TCustomIniFile.ReadBool (1281), TCustomIniFile.WriteBool (1281), TCustomIniFile.Options (1287)

41.4.41 TCustomIniFile.BoolFalseStrings

Synopsis: Strings to recognize as boolean `False` values.

Declaration: `Property BoolFalseStrings : TStringArray`

Visibility: public

Access: Read,Write

Description: `BoolFalseStrings` is a list of strings that will be recognized as boolean `False` value in TCustomIniFile.ReadBool (1281) The first string in the list will be used when writing a `False` boolean value in TCustomIniFile.WriteBool (1281).

The string values are ignored if `ifoWriteStringBoolean` is not in the ini file options (1287).

See also: TCustomIniFile.SetBoolStringValues (1279), TCustomIniFile.BoolTrueStrings (1288), TCustomIniFile.ReadBool (1281), TCustomIniFile.WriteBool (1281), TCustomIniFile.Options (1287)

41.4.42 TCustomIniFile.OwnsEncoding

Synopsis: Does the ini file instance own the encoding ?

Declaration: `Property OwnsEncoding : Boolean`

Visibility: public

Access: Read

Description: `OwnsEncoding` indicates whether the encoding is owned by the ini file instance or not. If it is owned, it will be freed on destroy. The value of this property is set in the constructor.

See also: TCustomIniFile.Encoding (1286), TCustomIniFile.Create (1278)

41.5 THashedStringList

41.5.1 Description

`THashedStringList` is a TStringList (**??**) descendent which creates hash values for the strings and names (in the case of a name-value pair) stored in it. The IndexOf (1290) and IndexOfName (1290) functions make use of these hash values to quickly locate a value.

See also: IndexOf (1290), IndexOfName (1290)

41.5.2 Method overview

Page	Method	Description
1289	Destroy	Clean up instance.
1290	IndexOf	Returns the index of a string in the list of strings.
1290	IndexOfName	Return the index of a name in the list of name=value pairs.

41.5.3 THashedStringList.Destroy

Synopsis: Clean up instance.

Declaration: `destructor Destroy;  Override`

Visibility: public

Description: `Destroy` cleans up the hash tables and then calls the inherited `Destroy`.

41.5.4 THashedStringList.IndexOf

Synopsis: Returns the index of a string in the list of strings.

Declaration: `function IndexOf(const S: string) : Integer;  Override`

Visibility: public

Description: `IndexOf` overrides the #rtl.classes.TStringList.IndexOf (**??**) method and uses the hash values to look for the location of `S`.

See also: #rtl.classes.TStringList.IndexOf (**??**), THashedStringList.IndexOfName (1290)

41.5.5 THashedStringList.IndexOfName

Synopsis: Return the index of a name in the list of name=value pairs.

Declaration: `function IndexOfName(const Name: string) : Integer;  Override`

Visibility: public

Description: `IndexOfName` overrides the #rtl.classes.TStrings.IndexOfName (**??**) method and uses the hash values of the names to look for the location of `Name`.

See also: #rtl.classes.TStrings.IndexOfName (**??**), THashedStringList.IndexOf (1290)

41.6 TIniFile

41.6.1 Description

`TIniFile` is an implementation of TCustomIniFile (1277) which does the same as TMemIniFile (1299), namely it reads the whole file into memory. Unlike `TMemIniFile` it does not cache updates in memory, but immediately writes any changes to disk.

`TIniFile` introduces no new methods, it just implements the abstract methods introduced in `TCustomIniFile`

See also: TCustomIniFile (1277), TMemIniFile (1299)

41.6.2 Method overview

Page	Method	Description
1291	Create	Create a new instance of `TIniFile`.
1293	DeleteKey	Delete key.
1291	Destroy	Remove the `TIniFile` instance from memory.
1293	EraseSection	
1292	ReadSection	Read the key names in a section.
1292	ReadSectionRaw	Read raw section.
1292	ReadSections	Read section names.
1293	ReadSectionValues	
1291	ReadString	Read a string.
1293	UpdateFile	Update the file on disk.
1292	WriteString	Write string to file.

41.6.3 Property overview

Page	Properties	Access	Description
1294	CacheUpdates	rw	Should changes be kept in memory.
1294	Stream	r	Stream from which ini file was read.
1294	WriteBOM	rw	Indicates if a Byte Order Mark (BOM) is written at the start of the .INI file.

41.6.4 TIniFile.Create

Synopsis: Create a new instance of `TIniFile`.

Declaration:
```
constructor Create(const AFileName: string; AOptions: TIniFileOptions)
                   ;  Override;  Overload
constructor Create(AStream: TStream; AOptions: TIniFileOptions)
                   ;  Overload
constructor Create(AStream: TStream; AEscapeLineFeeds: Boolean)
                   ;  Overload
constructor Create(AStream: TStream; ADefaultEncoding: TEncoding;
                   AOptions: TIniFileOptions)
constructor Create(AStream: TStream; ADefaultEncoding: TEncoding;
                   AOwnsEncoding: Boolean; AOptions: TIniFileOptions)
```

Visibility: public

Description: `Create` creates a new instance of `TIniFile` and initializes the class by reading the file from disk if the filename `AFileName` is specified, or from stream in case `AStream` is specified. It also sets most variables to their initial values, i.e. `AEscapeLineFeeds` is saved prior to reading the file, and `Cacheupdates` is set to `False`.

Default options for the TCustomIniFile.Options (1287) property can be specified in `AOptions`.

See also: TCustomIniFile (1277), TMemIniFile (1299), TCustomIniFile.Options (1287)

41.6.5 TIniFile.Destroy

Synopsis: Remove the `TIniFile` instance from memory.

Declaration:
```
destructor Destroy;  Override
```

Visibility: public

Description: `Destroy` writes any pending changes to disk, and cleans up the `TIniFile` structures, and then calls the inherited `Destroy`, effectively removing the instance from memory.

Errors: If an error happens when the file is written to disk, an exception will be raised.

See also: UpdateFile (1286), CacheUpdates (1294)

41.6.6 TIniFile.ReadString

Synopsis: Read a string.

Declaration:
```
function ReadString(const Section: string; const Ident: string;
                    const Default: string) : string;  Override
```

Visibility: public

Description: `ReadString` implements the TCustomIniFile.ReadString (1279) abstract method by looking at the in-memory copy of the ini file and returning the string found there.

See also: TCustomIniFile.ReadString (1279)

41.6.7 TIniFile.WriteString

Synopsis: Write string to file.

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

Visibility: public

Description: `WriteString` implements the TCustomIniFile.WriteString (1280) abstract method by writing the string to the in-memory copy of the ini file. If CacheUpdates (1294) property is `False`, then the whole file is immediately written to disk as well.

Errors: If an error happens when the file is written to disk, an exception will be raised.

41.6.8 TIniFile.ReadSection

Synopsis: Read the key names in a section.

Declaration:
```
procedure ReadSection(const Section: string; Strings: TStrings)
                      ;  Override
```

Visibility: public

Description: `ReadSection` reads the key names from `Section` into `Strings`, taking the in-memory copy of the ini file. This is the implementation for the abstract TCustomIniFile.ReadSection (1284)

See also: TCustomIniFile.ReadSection (1284), TIniFile.ReadSectionRaw (1292)

41.6.9 TIniFile.ReadSectionRaw

Synopsis: Read raw section.

Declaration:
```
procedure ReadSectionRaw(const Section: string; Strings: TStrings)
```

Visibility: public

Description: `ReadSectionRaw` returns the contents of the section `Section` as it is: this includes the comments in the section. (these are also stored in memory)

See also: TIniFile.ReadSection (1292), TCustomIniFile.ReadSection (1284)

41.6.10 TIniFile.ReadSections

Synopsis: Read section names.

Declaration:
```
procedure ReadSections(Strings: TStrings);  Override
```

Visibility: public

Description: `ReadSections` is the implementation of TCustomIniFile.ReadSections (1285). It operates on the in-memory copy of the inifile, and places all section names in `Strings`.

See also: TIniFile.ReadSection (1292), TCustomIniFile.ReadSections (1285), TIniFile.ReadSectionValues (1293)

41.6.11 TIniFile.ReadSectionValues

Synopsis:

Declaration:
```
procedure ReadSectionValues(const Section: string; Strings: TStrings;
                            AOptions: TSectionValuesOptions);  Override
                            ;  Overload
```

Visibility: public

Description: `ReadSectionValues` is the implementation of TCustomIniFile.ReadSectionValues (1285). It operates on the in-memory copy of the inifile, and places all key names from `Section` together with their values in `Strings`.

See also: TIniFile.ReadSection (1292), TCustomIniFile.ReadSectionValues (1285), TIniFile.ReadSections (1292)

41.6.12 TIniFile.EraseSection

Synopsis:

Declaration:
```
procedure EraseSection(const Section: string);  Override
```

Visibility: public

Description: `Erasesection` deletes the section `Section` from memory, if CacheUpdates (1294) is `False`, then the file is immediately updated on disk. This method is the implementation of the abstract TCustomIniFile.EraseSection (1285) method.

See also: TCustomIniFile.EraseSection (1285), TIniFile.ReadSection (1292), TIniFile.ReadSections (1292)

41.6.13 TIniFile.DeleteKey

Synopsis: Delete key.

Declaration:
```
procedure DeleteKey(const Section: string; const Ident: string)
                    ;  Override
```

Visibility: public

Description: `DeleteKey` deletes the `Ident` from the section `Section`. This operation is performed on the in-memory copy of the ini file. if CacheUpdates (1294) is `False`, then the file is immediately updated on disk.

See also: CacheUpdates (1294)

41.6.14 TIniFile.UpdateFile

Synopsis: Update the file on disk.

Declaration:
```
procedure UpdateFile;  Override
```

Visibility: public

Description: `UpdateFile` writes the in-memory data for the ini file to disk. The whole file is written. If the ini file was instantiated from a stream, then the stream is updated. Note that the stream must be seekable for this to work correctly. The ini file is marked as 'clean' after a call to `UpdateFile` (i.e. not in need of writing to disk).

Errors: If an error occurs when writing to stream or disk, an exception may be raised.

See also: CacheUpdates (1294)

41.6.15 TIniFile.Stream

Synopsis: Stream from which ini file was read.

Declaration: `Property Stream : TStream`

Visibility: public

Access: Read

Description: `Stream` is the stream which was used to create the `IniFile`. The UpdateFile (1293) method will use this stream to write changes to.

See also: Create (1291), UpdateFile (1293)

41.6.16 TIniFile.CacheUpdates

Synopsis: Should changes be kept in memory.

Declaration: `Property CacheUpdates : Boolean`

Visibility: public

Access: Read,Write

Description: `CacheUpdates` determines how to deal with changes to the ini-file data: if set to `True` then changes are kept in memory till the file is written to disk with a call to UpdateFile (1293). If it is set to `False` then each call that changes the data of the ini-file will result in a call to `UpdateFile`. This is the default behaviour, but it may adversely affect performance.

See also: UpdateFile (1293)

41.6.17 TIniFile.WriteBOM

Synopsis: Indicates if a Byte Order Mark (BOM) is written at the start of the .INI file.

Declaration: `Property WriteBOM : Boolean`

Visibility: public

Access: Read,Write

Description: `WriteBOM` is a `Boolean` property which indicates if a **Byte Order Mark** (**BOM**) is written at the start of the .INI file. The default value for the property is **False**, and causes the BOM to be omitted when storing the .INI file content.

Setting a new value for the property can cause the sections and section values to be re-written to the `FileName` or `Stream` where the .INI file content is stored. When `CacheUpdates` is **False**, the `UpdateFile` method is called to re-write the values in the storage. When CacheUpdates is **True**, the `Dirty` property is set to **True** and the action is deferred until the class instance is freed or update caching is disabled.

The property value is used in the `UpdatFile` method, and is assigned to the `WriteBOM` property in the `TStrings` instance used to write the content for the .INI file.

41.7 TIniFileKey

41.7.1 Description

`TIniFileKey` is used to keep the key/value pairs in the ini file in memory. It is an internal structure, used internally by the TIniFile (1290) class.

See also: TIniFile (1290)

41.7.2 Method overview

Page	Method	Description
1295	Create	Create a new instance of `TIniFileKey`.

41.7.3 Property overview

Page	Properties	Access	Description
1295	Ident	rw	Key name.
1295	Value	rw	Key value.

41.7.4 TIniFileKey.Create

Synopsis: Create a new instance of `TIniFileKey`.

Declaration: `constructor Create(const AIdent: string; const AValue: string)`

Visibility: public

Description: `Create` instantiates a new instance of `TIniFileKey` on the heap. It fills Ident (1295) with `AIdent` and Value (1295) with `AValue`.

See also: Ident (1295), Value (1295)

41.7.5 TIniFileKey.Ident

Synopsis: Key name.

Declaration: `Property Ident : string`

Visibility: public

Access: Read,Write

Description: `Ident` is the key value part of the key/value pair.

See also: Value (1295)

41.7.6 TIniFileKey.Value

Synopsis: Key value.

Declaration: `Property Value : string`

Visibility: public

Access: Read,Write

Description: `Value` is the value part of the key/value pair.

See also: Ident (1295)

41.8 TIniFileKeyList

41.8.1 Description

`TIniFileKeyList` maintains a list of TIniFileKey (1295) instances on behalf of the TIniFileSection (1297) class. It stores they keys of one section of the .ini files.

See also: TIniFileKey (1295), TIniFileSection (1297)

41.8.2 Method overview

Page	Method	Description
1296	Clear	Clear the list.
1296	Destroy	Free the instance.

41.8.3 Property overview

Page	Properties	Access	Description
1296	Items	r	Indexed access to `TIniFileKey` items in the list.

41.8.4 TIniFileKeyList.Destroy

Synopsis: Free the instance.

Declaration:
```
destructor Destroy;  Override
```

Visibility: public

Description: `Destroy` clears up the list using Clear (1296) and then calls the inherited destroy.

See also: Clear (1296)

41.8.5 TIniFileKeyList.Clear

Synopsis: Clear the list.

Declaration:
```
procedure Clear;  Override
```

Visibility: public

Description: `Clear` removes all TIniFileKey (1295) instances from the list, and frees the instances.

See also: TIniFileKey (1295)

41.8.6 TIniFileKeyList.Items

Synopsis: Indexed access to `TIniFileKey` items in the list.

Declaration:
```
Property Items[Index: Integer]: TIniFileKey; default
```

Visibility: public

Access: Read

Description: `Items` provides indexed access to the TIniFileKey (1295) items in the list. The index is zero-based and runs from 0 to `Count-1`.

See also: TIniFileKey (1295)

41.9 TIniFileSection

41.9.1 Description

`TIniFileSection` is a class which represents a section in the .ini, and is used internally by the TIniFile (1290) class (one instance of `TIniFileSection` is created for each section in the file by the TIniFileSectionList (1298) list). The name of the section is stored in the Name (1298) property, and the key/value pairs in this section are available in the KeyList (1298) property.

See also: TIniFileKeyList (1296), TIniFile (1290), TIniFileSectionList (1298)

41.9.2 Method overview

Page	Method	Description
1297	Create	Create a new section object.
1298	Destroy	Free the section object from memory.
1297	Empty	Is the section empty.

41.9.3 Property overview

Page	Properties	Access	Description
1298	KeyList	r	List of key/value pairs in this section.
1298	Name	r	Name of the section.

41.9.4 TIniFileSection.Empty

Synopsis: Is the section empty.

Declaration: `function Empty : Boolean`

Visibility: public

Description: `Empty` returns `True` if the section contains no key values (even if they are empty). It may contain comments.

41.9.5 TIniFileSection.Create

Synopsis: Create a new section object.

Declaration: `constructor Create(const AName: string)`

Visibility: public

Description: `Create` instantiates a new `TIniFileSection` class, and sets the name to `AName`. It allocates a TIniFileKeyList (1296) instance to keep all the key/value pairs for this section.

See also: TIniFileKeyList (1296)

41.9.6 TIniFileSection.Destroy

Synopsis: Free the section object from memory.

Declaration: `destructor Destroy; Override`

Visibility: public

Description: `Destroy` cleans up the key list, and then calls the inherited `Destroy`, removing the `TIniFileSection` instance from memory.

See also: Create (1297), TIniFileKeyList (1296)

41.9.7 TIniFileSection.Name

Synopsis: Name of the section.

Declaration: `Property Name : string`

Visibility: public

Access: Read

Description: `Name` is the name of the section in the file.

See also: TIniFileSection.KeyList (1298)

41.9.8 TIniFileSection.KeyList

Synopsis: List of key/value pairs in this section.

Declaration: `Property KeyList : TIniFileKeyList`

Visibility: public

Access: Read

Description: `KeyList` is the TIniFileKeyList (1296) instance that is used by the `TIniFileSection` to keep the key/value pairs of the section.

See also: TIniFileSection.Name (1298), TIniFileKeyList (1296)

41.10 TIniFileSectionList

41.10.1 Description

`TIniFileSectionList` maintains a list of TIniFileSection (1297) instances, one for each section in an .ini file. `TIniFileSectionList` is used internally by the TIniFile (1290) class to represent the sections in the file.

See also: TIniFileSection (1297), TIniFile (1290)

41.10.2 Method overview

Page	Method	Description
1299	Clear	Clear the list.
1299	Destroy	Free the object from memory.

41.10.3 Property overview

Page	Properties	Access	Description
1299	Items	r	Indexed access to all the section objects in the list.

41.10.4 TIniFileSectionList.Destroy

Synopsis: Free the object from memory.

Declaration: `destructor Destroy;  Override`

Visibility: public

Description: `Destroy` calls Clear (1299) to clear the section list and the calls the inherited `Destroy`

See also: Clear (1299)

41.10.5 TIniFileSectionList.Clear

Synopsis: Clear the list.

Declaration: `procedure Clear;  Override`

Visibility: public

Description: `Clear` removes all TIniFileSection (1297) items from the list, and frees the items it removes from the list.

See also: TIniFileSection (1297), TIniFileSectionList.Items (1299)

41.10.6 TIniFileSectionList.Items

Synopsis: Indexed access to all the section objects in the list.

Declaration: `Property Items[Index: Integer]: TIniFileSection; default`

Visibility: public

Access: Read

Description: `Items` provides indexed access to all the section objects in the list. `Index` should run from 0 to `Count-1`.

See also: TIniFileSection (1297), TIniFileSectionList.Clear (1299)

41.11 TMemIniFile

41.11.1 Description

`TMemIniFile` is a simple descendent of TIniFile (1290) which introduces some extra methods to be compatible to the Delphi implementation of `TMemIniFile`. The FPC implementation of `TIniFile` is implemented as a `TMemIniFile`, except that `TIniFile` does not cache its updates, and `TMemIniFile` does.

See also: TIniFile (1290), TCustomIniFile (1277), CacheUpdates (1294)

41.11.2 Method overview

Page	Method	Description
1300	Clear	Clear the data.
1300	Create	Create a new instance of `TMemIniFile`.
1300	GetStrings	Get contents of ini file as stringlist.
1301	Rename	Rename the ini file.
1301	SetStrings	Set data from a stringlist.

41.11.3 TMemIniFile.Create

Synopsis: Create a new instance of `TMemIniFile`.

Declaration:
```
constructor Create(const AFileName: string; AOptions: TIniFileOptions)
                   ; Override; Overload
constructor Create(const AFileName: string; AEscapeLineFeeds: Boolean)
                   ; Override; Overload
```

Visibility: public

Description: `Create` simply calls the inherited Create (1291), and sets the CacheUpdates (1294) to `True` so updates will be kept in memory till they are explicitly written to disk.

See also: TIniFile.Create (1291), CacheUpdates (1294)

41.11.4 TMemIniFile.Clear

Synopsis: Clear the data.

Declaration:
```
procedure Clear
```

Visibility: public

Description: `Clear` removes all sections and key/value pairs from memory. If CacheUpdates (1294) is set to `False` then the file on disk will immediately be emptied.

See also: SetStrings (1301), GetStrings (1300)

41.11.5 TMemIniFile.GetStrings

Synopsis: Get contents of ini file as stringlist.

Declaration:
```
procedure GetStrings(List: TStrings)
```

Visibility: public

Description: `GetStrings` returns the whole contents of the ini file in a single stringlist, `List`. This includes comments and empty sections.

The `GetStrings` call can be used to get data for a call to SetStrings (1301), which can be used to copy data between 2 in-memory ini files.

See also: SetStrings (1301), Clear (1300)

41.11.6 TMemIniFile.Rename

Synopsis: Rename the ini file.

Declaration: `procedure Rename(const AFileName: string; Reload: Boolean)`

Visibility: public

Description: `Rename` will rename the ini file with the new name `AFileName`. If `Reload` is `True` then the in-memory contents will be cleared and replaced with the contents found in `AFileName`, if it exists. If `Reload` is `False`, the next call to `UpdateFile` will replace the contents of `AFileName` with the in-memory data.

See also: UpdateFile (1293)

41.11.7 TMemIniFile.SetStrings

Synopsis: Set data from a stringlist.

Declaration: `procedure SetStrings(List: TStrings)`

Visibility: public

Description: `SetStrings` sets the in-memory data from the `List` stringlist. The data is first cleared.

The `SetStrings` call can be used to set the data of the ini file to a list of strings obtained with GetStrings (1300). The two calls combined can be used to copy data between 2 in-memory ini files.

See also: GetStrings (1300), Clear (1300)

41.12 TStringHash

41.12.1 Description

`TStringHash` is a Delphi compatibility object. It is not used in the TIniFile implementation. It implements a bucket list for `Name=Value` pairs, where `Value` is an integer. This enables quick lookup of values based on a name.

See also: TIniFIle (1290), TStringHash.Create (1302), TStringHash.ValueOf (1303)

41.12.2 Method overview

Page	Method	Description
1302	Add	Add a new value to the hash.
1302	Clear	Remove all values.
1302	Create	Create a new instance of `TStringHash`.
1302	Destroy	Free `TStringHash` instance.
1303	Modify	Try to modify an existing value.
1303	Remove	Remove a key from the hash.
1303	ValueOf	Retrieve value of `Key`.

41.12.3 Property overview

Page	Properties	Access	Description
1303	AddReplacesExisting	rw	Should `Add` replace existing values or not.

41.12.4 TStringHash.Create

Synopsis: Create a new instance of `TStringHash`.

Declaration: `constructor Create(ACapacity: Cardinal)`

Visibility: public

Description: `Create` instantiates a new instance of `TStringHash`. The `ACapacity` argument is present for Delphi compatibility, but is otherwise unused.

See also: TStringHash.ValueOf (1303), TStringHash.Destroy (1302)

41.12.5 TStringHash.Destroy

Synopsis: Free `TStringHash` instance.

Declaration: `destructor Destroy;  Override`

Visibility: public

Description: `Destroy` clears the internal data structures and removes the `TStringHash` instance from memory.

See also: TStringHash.Create (1302), TStringHash.Clear (1302)

41.12.6 TStringHash.Add

Synopsis: Add a new value to the hash.

Declaration: `procedure Add(const Key: string; Value: Integer)`

Visibility: public

Description: `Add` adds the value `Value` with name `AKey` to the list. The behaviour of `Add` depends on AddReplacesExisting (1303). If if it is `False`, then the existing value is left unchanged, and an exception is raised. If `AddReplacesExisting` is `True` then an existing value is replaced.

Errors: An exception will be raised if the value already exists and `AddReplacesExisting` is `False`

See also: TStringHash.AddReplacesExisting (1303), TStringHash.Modify (1303), TStringHash.Remove (1303), TStringHash.ValueOf (1303)

41.12.7 TStringHash.Clear

Synopsis: Remove all values.

Declaration: `procedure Clear`

Visibility: public

Description: `Clear` removes all values from the hash.

See also: TStringHash.Destroy (1302), TStringHash.Add (1302), TStringHash.Remove (1303), TStringHash.ValueOf (1303)

41.12.8 TStringHash.Modify

Synopsis: Try to modify an existing value.

Declaration: `function Modify(const Key: string; Value: Integer) : Boolean`

Visibility: public

Description: `Modify` will replace the value of `Key` with `Value`. `Key` must exist. It returns `True` if the operation was successful. If the value didn't exist, `False` is returned.

See also: TStringHash.Add (1302), TStringHash.Clear (1302), TStringHash.Remove (1303), TStringHash.ValueOf (1303)

41.12.9 TStringHash.Remove

Synopsis: Remove a key from the hash.

Declaration: `procedure Remove(const Key: string)`

Visibility: public

Description: `Remove` removes the key `Key` from the hash, if it was present.

Errors: None.

See also: TStringHash.Add (1302), TStringHash.Clear (1302), TStringHash.Modify (1303), TStringHash.ValueOf (1303)

41.12.10 TStringHash.ValueOf

Synopsis: Retrieve value of `Key`.

Declaration: `function ValueOf(const Key: string) : Integer`

Visibility: public

Description: `ValueOf` returns the value of `AKey`, if it is present. if the key is not present, `-1` is returned.

Errors: None.

See also: TStringHash.Add (1302), TStringHash.Clear (1302), TStringHash.Modify (1303), TStringHash.Remove (1303)

41.12.11 TStringHash.AddReplacesExisting

Synopsis: Should `Add` replace existing values or not.

Declaration: `Property AddReplacesExisting : Boolean`

Visibility: public

Access: Read,Write

Description: `AddReplacesExisting` indicates whether TStringHash.Add (1302) will replace an existing value (`True`) or will raise an exception when an existing value is added (`False`).

See also: TStringHash.Add (1302)

Chapter 42

Reference for unit 'iostream'

42.1 Used units

Table 42.1: Used units by unit 'iostream'

Name	Page
Classes	**??**
System	**??**

42.2 Overview

The iostream implements a descendent of THandleStream (**??**) streams that can be used to read from standard input and write to standard output and standard diagnostic output (`stderr`).

42.3 Constants, types and variables

42.3.1 Types

```
TIOSType = (iosInput,iosOutPut,iosError)
```

Table 42.2: Enumeration values for type TIOSType

Value	Explanation
iosError	The stream can be used to write to standard diagnostic output.
iosInput	The stream can be used to read from standard input.
iosOutPut	The stream can be used to write to standard output.

`TIOSType` is passed to the Create (1305) constructor of TIOStream (1305), it determines what kind of stream is created.

42.4 EIOStreamError

42.4.1 Description

Error thrown in case of an invalid operation on a TIOStream (1305).

42.5 TIOStream

42.5.1 Description

`TIOStream` can be used to create a stream which reads from or writes to the standard input, output or stderr file descriptors. It is a descendent of `THandleStream`. The type of stream that is created is determined by the TIOSType (1304) argument to the constructor. The handle of the standard input, output or stderr file descriptors is determined automatically.

The `TIOStream` keeps an internal `Position`, and attempts to provide minimal Seek (1306) behaviour based on this position.

See also: TIOSType (1304), THandleStream (**??**)

42.5.2 Method overview

Page	Method	Description
1305	Create	Construct a new instance of TIOStream (1305).
1305	Read	Read data from the stream.
1306	Seek	Set the stream position.
1306	Write	Write data to the stream.

42.5.3 TIOStream.Create

Synopsis: Construct a new instance of TIOStream (1305).

Declaration: `constructor Create(aIOSType: TIOSType)`

Visibility: public

Description: `Create` creates a new instance of TIOStream (1305), which can subsequently be used

Errors: No checking is performed to see whether the requested file descriptor is actually open for reading/writing. In that case, subsequent calls to `Read` or `Write` or `seek` will fail.

See also: TIOStream.Read (1305), TIOStream.Write (1306)

42.5.4 TIOStream.Read

Synopsis: Read data from the stream.

Declaration: `function Read(var Buffer; Count: LongInt) : LongInt;  Override`

Visibility: public

Description: `Read` checks first whether the type of the stream allows reading (type is `iosInput`). If not, it raises a EIOStreamError (1305) exception. If the stream can be read, it calls the inherited `Read` to actually read the data.

Errors: An `EIOStreamError` exception is raised if the stream does not allow reading.

See also: TIOSType (1304), TIOStream.Write (1306)

42.5.5 TIOStream.Write

Synopsis: Write data to the stream.

Declaration:
```
function Write(const Buffer; Count: LongInt) : LongInt;  Override
```

Visibility: public

Description: `Write` checks first whether the type of the stream allows writing (type is `iosOutput` or `iosError`). If not, it raises a EIOStreamError (1305) exception. If the stream can be written to, it calls the inherited `Write` to actually read the data.

Errors: An `EIOStreamError` exception is raised if the stream does not allow writing.

See also: TIOSType (1304), TIOStream.Read (1305)

42.5.6 TIOStream.Seek

Synopsis: Set the stream position.

Declaration:
```
function Seek(const Offset: Int64; Origin: TSeekOrigin) : Int64
                  ;  Override
```

Visibility: public

Description: `Seek` overrides the standard `Seek` implementation. Normally, standard input, output and stderr are not seekable. The `TIOStream` stream tries to provide seek capabilities for the following limited number of cases:

Origin=soFromBeginning If `Offset` is larger than the current position, then the remaining bytes are skipped by reading them from the stream and discarding them, if the stream is of type `iosInput`.

Origin=soFromCurrent If `Offset` is zero, the current position is returned. If it is positive, then `Offset` bytes are skipped by reading them from the stream and discarding them, if the stream is of type `iosInput`.

All other cases will result in a `EIOStreamError` exception.

Errors: An EIOStreamError (1305) exception is raised if the stream does not allow the requested seek operation.

See also: EIOStreamError (1305)

Chapter 43

Reference for unit 'json2yaml'

43.1 Used units

Table 43.1: Used units by unit 'json2yaml'

Name	Page
Classes	**??**
fpjson	826
System	**??**

43.2 Overview

This unit provides functionality to convert JSON data structures into YAML format. It implements the TJSONToYaml (1307) class which can convert any JSON data type (objects, arrays, strings, numbers, booleans, and null values) into equivalent YAML representation with proper indentation and formatting.

43.3 TJSONToYaml

43.3.1 Description

`TJSONToYaml` provides functionality to convert JSON data structures into YAML format. The class can handle all JSON data types including objects, arrays, strings, numbers, booleans, and null values. The conversion preserves the hierarchical structure and produces properly indented YAML output.

The class supports configurable indentation size and can optionally add a YAML document header (—). Conversion is performed by calling the Convert (1308) method with the source JSON data and target output stream.

See also: Convert (1308), AddHeader (1308), IndentSize (1308)

43.3.2 Method overview

Page	Method	Description
1308	Convert	Convert JSON data to YAML format

43.3.3 Property overview

Page	Properties	Access	Description
1308	AddHeader	rw	Controls whether to add YAML document header
1308	IndentSize	rw	Number of spaces used for indentation

43.3.4 TJSONToYaml.Convert

Synopsis: Convert JSON data to YAML format

Declaration: `procedure Convert(JSON: TJSONData; aOutput: TStream)`

Visibility: public

Description: The `Convert` method converts the provided JSON data structure to YAML format and writes the result to the specified output stream. The method handles all JSON data types: objects, arrays, strings, numbers, booleans, and null values.

The conversion process uses the current IndentSize (1308) setting for proper YAML indentation. If AddHeader (1308) is `True`, a YAML document separator (—) is written at the beginning of the output.

Objects are converted to YAML mappings with key-value pairs, arrays become YAML sequences with list items prefixed by dashes, and scalar values are converted to their string representations.

See also: AddHeader (1308), IndentSize (1308)

43.3.5 TJSONToYaml.AddHeader

Synopsis: Controls whether to add YAML document header

Declaration: `Property AddHeader : Boolean`

Visibility: public

Access: Read,Write

Description: The `AddHeader` property determines whether a YAML document separator (—) is written at the beginning of the converted output. When set to `True`, the YAML document header is added before the actual content. When `False`, only the converted JSON content is written to the output stream.

The YAML document separator is part of the YAML specification and indicates the start of a YAML document, which can be useful when multiple YAML documents are concatenated or when strict YAML compliance is required.

See also: Convert (1308)

43.3.6 TJSONToYaml.IndentSize

Synopsis: Number of spaces used for indentation

Declaration: `Property IndentSize : Integer`

Visibility: public

Access: Read,Write

Description: The `IndentSize` property specifies the number of spaces used for each level of indentation in the generated YAML output. This value controls the visual structure and readability of the converted YAML document.

If not explicitly set, the default indentation size of 2 spaces is used during conversion. The indentation is applied consistently throughout the document to maintain proper YAML structure for nested objects and arrays.

See also: Convert (1308)

Chapter 44

Reference for unit 'jsonini'

44.1 Used units

Table 44.1: Used units by unit 'jsonini'

Name	Page
Classes	??
DateUtils	??
fpjson	826
IniFiles	1275
jsonparser	1321
jsonscanner	1344
System	??
sysutils	??

44.2 Overview

This unit provides TJSONIniFile (1312), a class that implements the ini file interface using JSON format instead of traditional ini files. It allows applications to store configuration data as JSON objects while maintaining compatibility with the standard `TCustomIniFile` interface.

44.3 Basic usage of the JSON ini file class

`TJSONIniFile` provides a JSON-based alternative to traditional ini files while maintaining the familiar `TCustomIniFile` interface. Configuration data is stored as JSON objects where sections become JSON objects and keys within sections become properties of those objects.

The following example demonstrates basic reading and writing operations:

```
program JsonIniExample;

uses
  JsonIni, SysUtils;
```

```
var
  IniFile: TJSONIniFile;
begin
  // Create or open a JSON ini file
  IniFile := TJSONIniFile.Create('config.json');
  try
    // Write configuration values
    IniFile.WriteString('Database', 'Host', 'localhost');
    IniFile.WriteInteger('Database', 'Port', 5432);
    IniFile.WriteBool('Database', 'UseSSL', True);

    IniFile.WriteString('UI', 'Theme', 'Dark');
    IniFile.WriteInteger('UI', 'Width', 1024);
    IniFile.WriteInteger('UI', 'Height', 768);

    IniFile.UpdateFile;

    // Read configuration values with defaults
    WriteLn('Database Host: ', IniFile.ReadString('Database', 'Host', 'unknown'));
    WriteLn('Database Port: ', IniFile.ReadInteger('Database', 'Port', 0));
    WriteLn('Use SSL: ', IniFile.ReadBool('Database', 'UseSSL', False));

    WriteLn('UI Theme: ', IniFile.ReadString('UI', 'Theme', 'Light'));
    WriteLn('Window Size: ', IniFile.ReadInteger('UI', 'Width', 800), 'x',
            IniFile.ReadInteger('UI', 'Height', 600));

  finally
    IniFile.Free;
  end;
end.
```

The resulting JSON file structure looks like this:

```
{
  "Database": {
    "Host": "localhost",
    "Port": 5432,
    "UseSSL": true
  },
  "UI": {
    "Theme": "Dark",
    "Width": 1024,
    "Height": 768
  }
}
```

For applications that need to control when changes are written to disk, use the `CacheUpdates` property:

```
var
  IniFile: TJSONIniFile;
begin
  IniFile := TJSONIniFile.Create('config.json');
```

```
  try
    // Cache changes in memory
    IniFile.CacheUpdates := True;

    // Make multiple changes
    IniFile.WriteString('Section1', 'Key1', 'Value1');
    IniFile.WriteInteger('Section1', 'Key2', 42);
    IniFile.WriteString('Section2', 'Key1', 'Value2');

    // Write all changes to file at once
    IniFile.UpdateFile;

  finally
    IniFile.Free;
  end;
end.
```

To convert existing traditional ini files to JSON format, use the `ConvertIni` class method:

```
// Convert with type detection
TJSONIniFile.ConvertIni('old.ini', 'new.json', False);

// Convert keeping everything as strings
TJSONIniFile.ConvertIni('old.ini', 'new.json', True);
```

See also: TCustomIniFile (1310), TMemIniFile (1310), TJSONIniFile (1312)

44.4 TJSONIniFile

44.4.1 Description

`TJSONIniFile` is a descendant of TCustomIniFile (1310) that stores configuration data in JSON format instead of traditional ini file format. The JSON structure uses objects for sections and key-value pairs within those objects for the actual configuration entries. This allows for more flexible data storage while maintaining full compatibility with the standard ini file interface.

See also: TCustomIniFile (1310), TMemIniFile (1310)

44.4.2 Method overview

Page	Method	Description
1314	ConvertIni	Converts a traditional ini file to JSON format
1313	Create	Creates a new TJSONIniFile instance
1319	DeleteKey	Removes a key from a section
1314	Destroy	Destroys the TJSONIniFile instance
1319	EraseSection	Removes an entire section from the JSON ini file
1315	ReadBool	Reads a boolean value from the JSON ini file
1315	ReadDate	Reads a date value from the JSON ini file
1315	ReadDateTime	Reads a date-time value from the JSON ini file
1316	ReadFloat	Reads a floating-point value from the JSON ini file
1315	ReadInt64	Reads a 64-bit integer value from the JSON ini file
1314	ReadInteger	Reads an integer value from the JSON ini file
1318	ReadSection	Reads all key names from a section
1318	ReadSections	Reads all section names from the JSON ini file
1319	ReadSectionValues	Reads all key-value pairs from a section
1314	ReadString	Reads a string value from the JSON ini file
1316	ReadTime	Reads a time value from the JSON ini file
1319	UpdateFile	Writes the JSON data to file
1318	WriteBool	Writes a boolean value to the JSON ini file
1316	WriteDate	Writes a date value to the JSON ini file
1317	WriteDateTime	Writes a date-time value to the JSON ini file
1317	WriteFloat	Writes a floating-point value to the JSON ini file
1318	WriteInt64	Writes a 64-bit integer value to the JSON ini file
1317	WriteInteger	Writes an integer value to the JSON ini file
1316	WriteString	Writes a string value to the JSON ini file
1317	WriteTime	Writes a time value to the JSON ini file

44.4.3 Property overview

Page	Properties	Access	Description
1320	CacheUpdates	rw	Controls when changes are written to file
1320	Stream	r	Read-only access to the internal stream

44.4.4 TJSONIniFile.Create

Synopsis: Creates a new TJSONIniFile instance

Declaration:
```
constructor Create(const AFileName: string; AOptions: TIniFileOptions)
                   ; Override; Overload
constructor Create(AStream: TStream; AOptions: TJSONOptions); Overload
```

Visibility: public

Description: `Create` creates a new `TJSONIniFile` instance. The constructor has two overloaded forms:

- `Create(AFileName, AOptions)` - Creates an instance that reads from and writes to the specified file
- `Create(AStream, AOptions)` - Creates an instance that reads from the provided stream

If the file specified in `AFileName` does not exist, a new empty JSON object is created. If the file exists, it is parsed as JSON and loaded into memory.

See also: UpdateFile (1319), TCustomIniFile.Create (1310)

44.4.5 TJSONIniFile.Destroy

Synopsis: Destroys the TJSONIniFile instance

Declaration:
```
destructor Destroy;  Override
```

Visibility: public

Description: `Destroy` frees all allocated resources and destroys the `TJSONIniFile` instance. The internal JSON object is freed and the inherited destructor is called.

See also: Create (1313)

44.4.6 TJSONIniFile.ConvertIni

Synopsis: Converts a traditional ini file to JSON format

Declaration:
```
class procedure ConvertIni(const AIniFile: string;
                           const AJSONFile: string; StringsOnly: Boolean)
```

Visibility: public

Description: `ConvertIni` is a class method that converts an existing traditional ini file to JSON format. The method reads all sections and keys from the source ini file and writes them to a new JSON ini file. If `StringsOnly` is `True`, all values are stored as strings. If `StringsOnly` is `False`, the method attempts to detect the data type and store integers, floats, booleans, and dates in their appropriate JSON types.

See also: Create (1313), TMemIniFile (1310)

44.4.7 TJSONIniFile.ReadString

Synopsis: Reads a string value from the JSON ini file

Declaration:
```
function ReadString(const Section: string; const Ident: string;
                    const Default: string) : string;  Override
```

Visibility: public

Description: `ReadString` reads a string value from the specified section and key. If the key is not found, the `Default` value is returned. For structured JSON values (objects or arrays), the JSON representation is returned as a string. For other values, the string representation is returned.

See also: WriteString (1316), TCustomIniFile.ReadString (1310)

44.4.8 TJSONIniFile.ReadInteger

Synopsis: Reads an integer value from the JSON ini file

Declaration:
```
function ReadInteger(const Section: string; const Ident: string;
                     Default: LongInt) : LongInt;  Override
```

Visibility: public

Description: `ReadInteger` reads an integer value from the specified section and key. If the stored value is a JSON number, it is converted to integer. If the stored value is a string, it attempts to parse it as an integer. If the key is not found or conversion fails, the `Default` value is returned.

See also: WriteInteger (1317), ReadInt64 (1315)

44.4.9 TJSONIniFile.ReadInt64

Synopsis: Reads a 64-bit integer value from the JSON ini file

Declaration:
```
function ReadInt64(const Section: string; const Ident: string;
                   Default: Int64) : Int64;  Override
```

Visibility: public

Description: `ReadInt64` reads a 64-bit integer value from the specified section and key. If the stored value is a JSON number, it is converted to `Int64`. If the stored value is a string, it attempts to parse it as a 64-bit integer. If the key is not found or conversion fails, the `Default` value is returned.

See also: WriteInt64 (1318), ReadInteger (1314)

44.4.10 TJSONIniFile.ReadBool

Synopsis: Reads a boolean value from the JSON ini file

Declaration:
```
function ReadBool(const Section: string; const Ident: string;
                  Default: Boolean) : Boolean;  Override
```

Visibility: public

Description: `ReadBool` reads a boolean value from the specified section and key. If the stored value is a JSON boolean, it is returned directly. For other value types, the method attempts to convert them to boolean. If the key is not found or conversion fails, the `Default` value is returned.

See also: WriteBool (1318)

44.4.11 TJSONIniFile.ReadDate

Synopsis: Reads a date value from the JSON ini file

Declaration:
```
function ReadDate(const Section: string; const Ident: string;
                  Default: TDateTime) : TDateTime;  Override
```

Visibility: public

Description: `ReadDate` reads a date value from the specified section and key. If the stored value is a JSON number, it is interpreted as a `TDateTime` value. If the stored value is a string, it is parsed using the ISO date format (yyyy-mm-dd). If the key is not found or conversion fails, the `Default` value is returned.

See also: WriteDate (1316), ReadDateTime (1315), ReadTime (1316)

44.4.12 TJSONIniFile.ReadDateTime

Synopsis: Reads a date-time value from the JSON ini file

Declaration:
```
function ReadDateTime(const Section: string; const Ident: string;
                      Default: TDateTime) : TDateTime;  Override
```

Visibility: public

Description: `ReadDateTime` reads a date-time value from the specified section and key. If the stored value is a JSON number, it is interpreted as a `TDateTime` value. If the stored value is a string, it is parsed using ISO date-time format (yyyy-mm-ddThh:nn:ss.zzz). If the key is not found or conversion fails, the `Default` value is returned.

See also: WriteDateTime (1317), ReadDate (1315), ReadTime (1316)

44.4.13 TJSONIniFile.ReadFloat

Synopsis: Reads a floating-point value from the JSON ini file

Declaration:
```
function ReadFloat(const Section: string; const Ident: string;
                   Default: Double) : Double;  Override
```

Visibility: public

Description: `ReadFloat` reads a floating-point value from the specified section and key. If the stored value is a JSON number, it is converted to `Double`. If the stored value is a string, it attempts to parse it as a floating-point number using both localized and non-localized formats. If the key is not found or conversion fails, the `Default` value is returned.

See also: WriteFloat (1317)

44.4.14 TJSONIniFile.ReadTime

Synopsis: Reads a time value from the JSON ini file

Declaration:
```
function ReadTime(const Section: string; const Ident: string;
                  Default: TDateTime) : TDateTime;  Override
```

Visibility: public

Description: `ReadTime` reads a time value from the specified section and key. If the stored value is a JSON number, the fractional part is extracted as the time component. If the stored value is a string, it is parsed using time format (0000-00-00Thh:nn:ss.zzz). If the key is not found or conversion fails, the `Default` value is returned.

See also: WriteTime (1317), ReadDate (1315), ReadDateTime (1315)

44.4.15 TJSONIniFile.WriteString

Synopsis: Writes a string value to the JSON ini file

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

Visibility: public

Description: `WriteString` writes a string value to the specified section and key. The value is stored as a JSON string in the underlying JSON object. If the section does not exist, it is created automatically.

See also: ReadString (1314)

44.4.16 TJSONIniFile.WriteDate

Synopsis: Writes a date value to the JSON ini file

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

Visibility: public

Description: `WriteDate` writes a date value to the specified section and key. The date is formatted as an ISO date string (yyyy-mm-ddT00:00:00.zzz) and stored as a JSON string. If the section does not exist, it is created automatically.

See also: ReadDate (1315), WriteDateTime (1317), WriteTime (1317)

44.4.17 TJSONIniFile.WriteDateTime

Synopsis: Writes a date-time value to the JSON ini file

Declaration:
```
procedure WriteDateTime(const Section: string; const Ident: string;
                        Value: TDateTime);  Override
```

Visibility: public

Description: `WriteDateTime` writes a date-time value to the specified section and key. The date-time is formatted as an ISO date-time string (yyyy-mm-ddThh:nn:ss.zzz) and stored as a JSON string. If the section does not exist, it is created automatically.

See also: ReadDateTime (1315), WriteDate (1316), WriteTime (1317)

44.4.18 TJSONIniFile.WriteFloat

Synopsis: Writes a floating-point value to the JSON ini file

Declaration:
```
procedure WriteFloat(const Section: string; const Ident: string;
                     Value: Double);  Override
```

Visibility: public

Description: `WriteFloat` writes a floating-point value to the specified section and key. The value is stored as a JSON number in the underlying JSON object. If the section does not exist, it is created automatically.

See also: ReadFloat (1316)

44.4.19 TJSONIniFile.WriteTime

Synopsis: Writes a time value to the JSON ini file

Declaration:
```
procedure WriteTime(const Section: string; const Ident: string;
                    Value: TDateTime);  Override
```

Visibility: public

Description: `WriteTime` writes a time value to the specified section and key. The time is formatted as a string (0000-00-00Thh:nn:ss.zzz) and stored as a JSON string. If the section does not exist, it is created automatically.

See also: ReadTime (1316), WriteDate (1316), WriteDateTime (1317)

44.4.20 TJSONIniFile.WriteInteger

Synopsis: Writes an integer value to the JSON ini file

Declaration:
```
procedure WriteInteger(const Section: string; const Ident: string;
                       Value: LongInt);  Override
```

Visibility: public

Description: `WriteInteger` writes an integer value to the specified section and key. The value is stored as a JSON number in the underlying JSON object. If the section does not exist, it is created automatically.

See also: ReadInteger (1314), WriteInt64 (1318)

44.4.21 TJSONIniFile.WriteInt64

Synopsis: Writes a 64-bit integer value to the JSON ini file

Declaration:
```
procedure WriteInt64(const Section: string; const Ident: string;
                     Value: Int64);  Override
```

Visibility: public

Description: `WriteInt64` writes a 64-bit integer value to the specified section and key. The value is stored as a JSON number in the underlying JSON object. If the section does not exist, it is created automatically.

See also: ReadInt64 (1315), WriteInteger (1317)

44.4.22 TJSONIniFile.WriteBool

Synopsis: Writes a boolean value to the JSON ini file

Declaration:
```
procedure WriteBool(const Section: string; const Ident: string;
                    Value: Boolean);  Override
```

Visibility: public

Description: `WriteBool` writes a boolean value to the specified section and key. The value is stored as a JSON boolean (`True` or `False`) in the underlying JSON object. If the section does not exist, it is created automatically.

See also: ReadBool (1315)

44.4.23 TJSONIniFile.ReadSection

Synopsis: Reads all key names from a section

Declaration:
```
procedure ReadSection(const Section: string; Strings: TStrings)
                      ;  Override
```

Visibility: public

Description: `ReadSection` reads all key names from the specified section and adds them to the `Strings` list. Only keys with actual values (not JSON objects or arrays) are included in the list. If the section does not exist, the list remains unchanged.

See also: ReadSections (1318), ReadSectionValues (1319)

44.4.24 TJSONIniFile.ReadSections

Synopsis: Reads all section names from the JSON ini file

Declaration:
```
procedure ReadSections(Strings: TStrings);  Override
```

Visibility: public

Description: `ReadSections` reads all section names from the JSON ini file and adds them to the `Strings` list. Only top-level JSON objects are considered sections.

See also: ReadSection (1318), ReadSectionValues (1319)

44.4.25 TJSONIniFile.ReadSectionValues

Synopsis: Reads all key-value pairs from a section

Declaration:
```
procedure ReadSectionValues(const Section: string; Strings: TStrings;
                            AOptions: TSectionValuesOptions);  Override
                            ;  Overload
```

Visibility: public

Description: `ReadSectionValues` reads all key-value pairs from the specified section and adds them to the `Strings` list in the format "key=value". The `AOptions` parameter controls which types of values are included. If `svoIncludeInvalid` is in `AOptions`, structured JSON values (objects and arrays) are included as their JSON representation.

See also: ReadSection (1318), ReadSections (1318)

44.4.26 TJSONIniFile.EraseSection

Synopsis: Removes an entire section from the JSON ini file

Declaration:
```
procedure EraseSection(const Section: string);  Override
```

Visibility: public

Description: `EraseSection` removes the specified section and all its key-value pairs from the JSON ini file. If the section does not exist, the method has no effect. The file is automatically updated unless `CacheUpdates` is `True`.

See also: DeleteKey (1319)

44.4.27 TJSONIniFile.DeleteKey

Synopsis: Removes a key from a section

Declaration:
```
procedure DeleteKey(const Section: string; const Ident: string)
                    ;  Override
```

Visibility: public

Description: `DeleteKey` removes the specified key from the specified section. If the section or key does not exist, the method has no effect. The file is automatically updated unless `CacheUpdates` is `True`.

See also: EraseSection (1319)

44.4.28 TJSONIniFile.UpdateFile

Synopsis: Writes the JSON data to file

Declaration:
```
procedure UpdateFile;  Override;  Overload
procedure UpdateFile(const AFileName: string);  Overload
```

Visibility: public

Description: `UpdateFile` writes the current JSON data to the file. The method has two overloaded forms:

- `UpdateFile()` - writes to the file specified in the constructor
- `UpdateFile(AFileName)` - writes to the specified file

The JSON data is formatted with proper indentation before writing.

See also: Create (1313), CacheUpdates (1320)

44.4.29 TJSONIniFile.Stream

Synopsis: Read-only access to the internal stream

Declaration: `Property Stream : TStream`

Visibility: public

Access: Read

Description: `Stream` provides read-only access to the internal stream used by the `TJSONIniFile` instance. This property is only available when the instance was created with a stream parameter. For file-based instances, this property returns `nil`.

See also: Create (1313)

44.4.30 TJSONIniFile.CacheUpdates

Synopsis: Controls when changes are written to file

Declaration: `Property CacheUpdates : Boolean`

Visibility: public

Access: Read,Write

Description: `CacheUpdates` controls when changes to the JSON data are written to the file. When `CacheUpdates` is `True`, changes are cached in memory and only written to file when `UpdateFile` is called explicitly or when `CacheUpdates` is set to `False`. When `CacheUpdates` is `False`, changes are written to file immediately.

See also: UpdateFile (1319)

Chapter 45

Reference for unit 'jsonparser'

45.1 Used units

Table 45.1: Used units by unit 'jsonparser'

Name	Page
Classes	??
fpjson	826
jsonreader	1323
jsonscanner	1344
System	??
sysutils	??

45.2 Overview

The `jsonparser` unit provides a parser for data encoded in JSON (JavaScript Object Notation) format. It implements `TJSONParser` class that can parse JSON strings and streams into `TJSONData` objects.

45.3 Constants, types and variables

45.3.1 Types

```
EJSONParser = jsonReader.EJSONParser
```

`EJSONParser` is an alias for the JSON parser exception class defined in the jsonreader unit. It is raised when syntax errors or other parsing problems are encountered during JSON parsing.

45.4 TJSONParser

45.4.1 Description

`TJSONParser` is a parser class for reading JSON data from streams or strings. It extends `TBaseJSONReader` and builds a tree of `TJSONData` objects representing the parsed JSON structure.

See also: TJSONData (1321), TBaseJSONReader (1321)

45.4.2 Method overview

Page	Method	Description
1322	Parse	Parses JSON data and returns root object

45.4.3 TJSONParser.Parse

Synopsis: Parses JSON data and returns root object

Declaration: `function Parse : TJSONData`

Visibility: public

Description: `Parse` reads and parses JSON data from the input stream, building a complete tree of JSON objects. It returns the root `TJSONData` object representing the entire parsed JSON structure.

See also: TJSONData (1321)

Chapter 46

Reference for unit 'jsonreader'

46.1 Used units

Table 46.1: Used units by unit 'jsonreader'

Name	Page
Classes	??
fpjson	826
jsonscanner	1344
System	??
sysutils	??

46.2 Overview

The `jsonreader` unit provides event-based JSON reading functionality. It offers a SAX-like approach to JSON parsing where events are fired as JSON elements are encountered during parsing. This allows for efficient processing of large JSON documents without loading the entire structure into memory. The unit includes abstract base classes, event-driven readers, and interface-based consumers for flexible JSON processing.

46.3 Using JSON readers with streams and files

Both TJSONEventReader (1338) and TJSONConsumerReader (1337) can read JSON data from various sources including streams, files, and strings.

This flexibility allows you to process JSON data from files, network streams, or any other stream-based source without loading the entire content into memory first.

Here's an example that reads JSON from a file stream:

```
program JSONStreamDemo;

{$mode objfpc}{$H+}

uses
```

```
  Classes, SysUtils, jsonreader;

type
  TJSONCounter = class
  private
    FObjectCount: Integer;
    FArrayCount: Integer;
    FStringCount: Integer;
    FNumberCount: Integer;
  public
    procedure OnStartObject(Sender: TObject);
    procedure OnStartArray(Sender: TObject);
    procedure OnStringValue(Sender: TObject; const AValue: UTF8String);
    procedure OnIntegerValue(Sender: TObject; const AValue: Integer);
    procedure OnFloatValue(Sender: TObject; const AValue: Double);

    property ObjectCount: Integer read FObjectCount;
    property ArrayCount: Integer read FArrayCount;
    property StringCount: Integer read FStringCount;
    property NumberCount: Integer read FNumberCount;
  end;

procedure TJSONCounter.OnStartObject(Sender: TObject);
begin
  Inc(FObjectCount);
end;

procedure TJSONCounter.OnStartArray(Sender: TObject);
begin
  Inc(FArrayCount);
end;

procedure TJSONCounter.OnStringValue(Sender: TObject; const AValue: UTF8String);
begin
  Inc(FStringCount);
end;

procedure TJSONCounter.OnIntegerValue(Sender: TObject; const AValue: Integer);
begin
  Inc(FNumberCount);
end;

procedure TJSONCounter.OnFloatValue(Sender: TObject; const AValue: Double);
begin
  Inc(FNumberCount);
end;

var
  FileStream: TFileStream;
  Reader: TJSONEventReader;
  Counter: TJSONCounter;
  JSONData: string;
  TempFile: TextFile;
begin
```

```
  // Create a sample JSON file
  JSONData := '{"users": [{"name": "John", "age": 25}, {"name": "Jane", "age": 30}],
  AssignFile(TempFile, '/tmp/claude/sample.json');
  Rewrite(TempFile);
  try
    Write(TempFile, JSONData);
  finally
    CloseFile(TempFile);
  end;

  Counter := TJSONCounter.Create;
  FileStream := TFileStream.Create('/tmp/claude/sample.json', fmOpenRead);
  Reader := TJSONEventReader.Create(FileStream, []);
  try
    // Assign counting event handlers
    Reader.OnStartObject := @Counter.OnStartObject;
    Reader.OnStartArray := @Counter.OnStartArray;
    Reader.OnStringValue := @Counter.OnStringValue;
    Reader.OnIntegerValue := @Counter.OnIntegerValue;
    Reader.OnFloatValue := @Counter.OnFloatValue;

    // Parse the JSON file
    Reader.Execute;

    // Display statistics
    Writeln('JSON Statistics:');
    Writeln('  Objects: ', Counter.ObjectCount);
    Writeln('  Arrays: ', Counter.ArrayCount);
    Writeln('  Strings: ', Counter.StringCount);
    Writeln('  Numbers: ', Counter.NumberCount);

  finally
    Reader.Free;
    FileStream.Free;
    Counter.Free;

    // Clean up temp file
    DeleteFile('/tmp/claude/sample.json');
  end;
end.
```

You can also use the readers with string sources when the JSON data is already in memory:

```
// String source
Reader := TJSONEventReader.Create(JSONString, []);

// Unicode string source
Reader := TJSONEventReader.Create(UnicodeJSONString, []);

// Stream source with options
Reader := TJSONEventReader.Create(MyStream, [joUTF8, joStrict]);
```

The parsing options (`TJSONOptions`) allow you to control:

- `joUTF8`: Handle UTF-8 encoding properly
- `joStrict`: Use strict JSON parsing rules
- `joIgnoreTrailingComma`: Allow trailing commas in objects and arrays
- `joComments`: Allow JavaScript-style comments in JSON

46.4 Using TJSONConsumerReader with IJSONConsumer interface

TJSONConsumerReader (1337) provides an alternative to event-based parsing by using the IJSONConsumer (1332) interface. Instead of assigning event handlers, you implement the interface methods to process JSON elements.

This approach provides a cleaner object-oriented design and can be easier to maintain for complex JSON processing logic.

The workflow involves:

1. Create a class that implements IJSONConsumer (1332)
2. Create a TJSONConsumerReader (1337) instance
3. Assign your consumer to the Consumer (1338) property
4. Call Execute (1338) to start parsing

Here's an example that builds a simple data structure from JSON:

```
program JSONConsumerDemo;

{$mode objfpc}{$H+}

uses
  Classes, SysUtils, jsonreader, Generics.Collections;

type
  TJSONDataExtractor = class(TInterfacedObject, IJSONConsumer)
  private
    FStack: TList<string>;
    FCurrentPath: string;
    FData: TStringList;
  public
    constructor Create;
    destructor Destroy; override;

    // IJSONConsumer interface
    procedure NullValue;
    procedure BooleanValue(const AValue: Boolean);
    procedure NumberValue(const AValue: string);
    procedure FloatValue(const AValue: Double);
    procedure Int64Value(const AValue: Int64);
    procedure QWordValue(const AValue: QWord);
    procedure IntegerValue(const AValue: Integer);
```

```
    procedure StringValue(const AValue: UTF8String);
    procedure KeyName(const AKey: UTF8String);
    procedure StartObject;
    procedure EndObject;
    procedure StartArray;
    procedure EndArray;

    property Data: TStringList read FData;
  end;

constructor TJSONDataExtractor.Create;
begin
  inherited Create;
  FStack := TList<string>.Create;
  FData := TStringList.Create;
end;

destructor TJSONDataExtractor.Destroy;
begin
  FStack.Free;
  FData.Free;
  inherited Destroy;
end;

procedure TJSONDataExtractor.KeyName(const AKey: UTF8String);
begin
  if FCurrentPath <> '' then
    FCurrentPath := FCurrentPath + '.' + AKey
  else
    FCurrentPath := AKey;
end;

procedure TJSONDataExtractor.StringValue(const AValue: UTF8String);
begin
  if FCurrentPath <> '' then
    FData.Values[FCurrentPath] := AValue;
  FCurrentPath := '';
end;

procedure TJSONDataExtractor.IntegerValue(const AValue: Integer);
begin
  if FCurrentPath <> '' then
    FData.Values[FCurrentPath] := IntToStr(AValue);
  FCurrentPath := '';
end;

procedure TJSONDataExtractor.BooleanValue(const AValue: Boolean);
begin
  if FCurrentPath <> '' then
    FData.Values[FCurrentPath] := BoolToStr(AValue, True);
  FCurrentPath := '';
end;

procedure TJSONDataExtractor.StartObject;
```

```
begin
  if FCurrentPath <> '' then
    FStack.Add(FCurrentPath);
end;

procedure TJSONDataExtractor.EndObject;
begin
  if FStack.Count > 0 then
  begin
    FCurrentPath := FStack[FStack.Count - 1];
    FStack.Delete(FStack.Count - 1);
  end
  else
    FCurrentPath := '';
end;

// Implement remaining interface methods
procedure TJSONDataExtractor.NullValue; begin FCurrentPath := ''; end;
procedure TJSONDataExtractor.NumberValue(const AValue: UTF8String); begin StringValu
procedure TJSONDataExtractor.FloatValue(const AValue: Double); begin StringValue(Flo
procedure TJSONDataExtractor.Int64Value(const AValue: Int64); begin StringValue(IntT
procedure TJSONDataExtractor.QWordValue(const AValue: QWord); begin StringValue(IntT
procedure TJSONDataExtractor.StartArray; begin end;
procedure TJSONDataExtractor.EndArray; begin end;

var
  Reader: TJSONConsumerReader;
  Extractor: TJSONDataExtractor;
  JSONData: string;
  i: Integer;
begin
  JSONData := '{"user": {"name": "Bob Jones", "age": 35, "active": true}, "settings"

  Extractor := TJSONDataExtractor.Create;
  Reader := TJSONConsumerReader.Create(JSONData, []);
  try
    Reader.Consumer := Extractor;
    Reader.Execute;

    Writeln('Extracted data:');
    for i := 0 to Extractor.Data.Count - 1 do
      Writeln('  ', Extractor.Data[i]);

  finally
    Reader.Free;
    Extractor.Free;
  end;
end.
```

The consumer interface approach is beneficial when you need to:

- Implement complex parsing logic in a structured way
- Reuse the same parsing logic across different parts of your application

- Build data structures incrementally during parsing
- Keep parsing state in a more organized manner

46.5 Using TJSONEventReader for event-based JSON parsing

TJSONEventReader (1338) provides event-based JSON parsing similar to SAX parsing for XML. As the parser encounters different JSON elements, it fires corresponding events that you can handle to process the data.

This approach is memory-efficient for large JSON documents since it doesn't load the entire structure into memory. You only need to handle the events for the data you're interested in processing.

Basic usage involves:

1. Create a TJSONEventReader (1338) instance with JSON source
2. Assign event handlers for the JSON elements you want to process
3. Call Execute (1339) to start parsing

Here's a complete example that parses JSON and extracts specific values:

```
program JSONEventReaderDemo;

{$mode objfpc}{$H+}

uses
  Classes, SysUtils, jsonreader;

type
  TMyJSONProcessor = class
  private
    FPersonName: string;
    FPersonAge: Integer;
    FInAddress: Boolean;
    FCurrentKey: string;
  public
    procedure OnKeyName(Sender: TObject; const AKey: UTF8String);
    procedure OnStringValue(Sender: TObject; const AValue: UTF8String);
    procedure OnIntegerValue(Sender: TObject; const AValue: Integer);
    procedure OnStartObject(Sender: TObject);
    procedure OnEndObject(Sender: TObject);

    property PersonName: string read FPersonName;
    property PersonAge: Integer read FPersonAge;
  end;

procedure TMyJSONProcessor.OnKeyName(Sender: TObject; const AKey: UTF8String);
begin
  FCurrentKey := AKey;
  FInAddress := (AKey = 'address');
end;
```

```
procedure TMyJSONProcessor.OnStringValue(Sender: TObject; const AValue: UTF8String);
begin
  if not FInAddress and (FCurrentKey = 'name') then
    FPersonName := AValue;
end;

procedure TMyJSONProcessor.OnIntegerValue(Sender: TObject; const AValue: Integer);
begin
  if not FInAddress and (FCurrentKey = 'age') then
    FPersonAge := AValue;
end;

procedure TMyJSONProcessor.OnStartObject(Sender: TObject);
begin
  // Object started
end;

procedure TMyJSONProcessor.OnEndObject(Sender: TObject);
begin
  FInAddress := False;
end;

var
  Reader: TJSONEventReader;
  Processor: TMyJSONProcessor;
  JSONData: string;
begin
  JSONData := '{"name": "Alice Smith", "age": 28, "address": {"street": "123 Oak St"

  Processor := TMyJSONProcessor.Create;
  Reader := TJSONEventReader.Create(JSONData, []);
  try
    // Assign event handlers
    Reader.OnKeyName := @Processor.OnKeyName;
    Reader.OnStringValue := @Processor.OnStringValue;
    Reader.OnIntegerValue := @Processor.OnIntegerValue;
    Reader.OnStartObject := @Processor.OnStartObject;
    Reader.OnEndObject := @Processor.OnEndObject;

    // Parse the JSON
    Reader.Execute;

    // Display results
    Writeln('Person Name: ', Processor.PersonName);
    Writeln('Person Age: ', Processor.PersonAge);

  finally
    Reader.Free;
    Processor.Free;
  end;
end.
```

The event-based approach is ideal when you need to:

- Process large JSON files without loading them entirely into memory
- Extract specific values from complex JSON structures
- Transform JSON data during parsing
- Handle JSON streams in real-time

46.6 Constants, types and variables

46.6.1 Types

```
TOnJSONBoolean = procedure(Sender: TObject; const AValue: Boolean
  )
                              of object
```

TOnJSONBoolean is an event handler type that is called when a JSON boolean value is encountered during parsing. The event provides the sender object and the boolean value that was found in the JSON data.

```
TOnJSONFloat = procedure(Sender: TObject; const AValue: TJSONFloat
  )
                            of object
```

TOnJSONFloat is an event handler type that is called when a JSON floating-point number value is encountered during parsing. The event provides the sender object and the floating-point value that was found in the JSON data.

```
TOnJSONInt64 = procedure(Sender: TObject; const AValue: Int64) of
  object
```

TOnJSONInt64 is an event handler type that is called when a JSON 64-bit integer value is encountered during parsing. The event provides the sender object and the 64-bit integer value that was found in the JSON data.

```
TOnJSONInteger = procedure(Sender: TObject; const AValue: Integer
  )
                              of object
```

TOnJSONInteger is an event handler type that is called when a JSON integer value is encountered during parsing. The event provides the sender object and the integer value that was found in the JSON data.

```
TOnJSONKey = procedure(Sender: TObject; const AKey: TJSONStringType
  )
                           of object
```

TOnJSONKey is an event handler type that is called when a JSON object key name is encountered during parsing. The event provides the sender object and the key name that was found in the JSON object data.

```
TOnJSONQWord = procedure(Sender: TObject; const AValue: QWord) of
  object
```

TOnJSONQWord is an event handler type that is called when a JSON unsigned quadword (64-bit unsigned integer) value is encountered during parsing. The event provides the sender object and the quadword value that was found in the JSON data.

```
TOnJSONString = procedure(Sender: TObject;
  const AValue: TJSONStringType) of object
```

TOnJSONString is an event handler type that is called when a JSON string value is encountered during parsing. The event provides the sender object and the string value that was found in the JSON data.

46.7 EJSONParser

46.7.1 Description

EJSONParser is raised when JSON parsing errors are encountered. It inherits from EParserError and provides specific error information for JSON syntax violations and parsing problems. The exception includes details about the location of the error in the JSON input, including line and column information.

See also: TBaseJSONReader (1336), TJSONEventReader (1338), TJSONConsumerReader (1337)

46.8 IJSONConsumer

46.8.1 Description

IJSONConsumer defines a standard interface for objects that consume JSON data during parsing. This interface provides methods that correspond to different JSON element types and structural events. Implementers of this interface can process JSON data as it is parsed without needing to create event handlers. The interface is used with TJSONConsumerReader (1337) for flexible JSON processing.

See also: TJSONConsumerReader (1337), TJSONEventReader (1338)

46.8.2 Method overview

Page	Method	Description
1333	BooleanValue	Processes a JSON boolean value
1336	EndArray	Processes the end of a JSON array
1335	EndObject	Processes the end of a JSON object
1333	FloatValue	Processes a JSON floating-point value
1334	Int64Value	Processes a JSON 64-bit integer value
1334	IntegerValue	Processes a JSON integer value
1335	KeyName	Processes a JSON object key name
1333	NullValue	Processes a JSON null value
1333	NumberValue	Processes a JSON number value as string
1334	QWordValue	Processes a JSON quadword value
1335	StartArray	Processes the start of a JSON array
1335	StartObject	Processes the start of a JSON object
1334	StringValue	Processes a JSON string value

46.8.3 IJSONConsumer.NullValue

Synopsis: Processes a JSON null value

Declaration: `procedure NullValue`

Visibility: default

Description: `NullValue` is called when a JSON null value is encountered during parsing. Implementers should handle the null value according to their processing requirements.

Errors: Implementation-specific error handling.

46.8.4 IJSONConsumer.BooleanValue

Synopsis: Processes a JSON boolean value

Declaration: `procedure BooleanValue(const AValue: Boolean)`

Visibility: default

Description: `BooleanValue` is called when a JSON boolean value is encountered during parsing. The method receives the boolean value that was found in the JSON data. Implementers should handle the boolean value according to their processing requirements.

Errors: Implementation-specific error handling.

46.8.5 IJSONConsumer.NumberValue

Synopsis: Processes a JSON number value as string

Declaration: `procedure NumberValue(const AValue: TJSONStringType)`

Visibility: default

Description: `NumberValue` is called when a JSON number value is encountered during parsing. The method receives the number as its string representation before type conversion. Implementers should handle the number string according to their processing requirements.

Errors: Implementation-specific error handling.

46.8.6 IJSONConsumer.FloatValue

Synopsis: Processes a JSON floating-point value

Declaration: `procedure FloatValue(const AValue: TJSONFloat)`

Visibility: default

Description: `FloatValue` is called when a JSON floating-point number is encountered during parsing. The method receives the floating-point value that was parsed from the JSON data. Implementers should handle the float value according to their processing requirements.

Errors: Implementation-specific error handling.

46.8.7 IJSONConsumer.Int64Value

Synopsis: Processes a JSON 64-bit integer value

Declaration: `procedure Int64Value(const AValue: Int64)`

Visibility: default

Description: `Int64Value` is called when a JSON 64-bit integer is encountered during parsing. The method receives the 64-bit integer value that was parsed from the JSON data. Implementers should handle the integer value according to their processing requirements.

Errors: Implementation-specific error handling.

46.8.8 IJSONConsumer.QWordValue

Synopsis: Processes a JSON quadword value

Declaration: `procedure QWordValue(const AValue: QWord)`

Visibility: default

Description: `QWordValue` is called when a JSON unsigned 64-bit integer is encountered during parsing. The method receives the quadword value that was parsed from the JSON data. Implementers should handle the quadword value according to their processing requirements.

Errors: Implementation-specific error handling.

46.8.9 IJSONConsumer.IntegerValue

Synopsis: Processes a JSON integer value

Declaration: `procedure IntegerValue(const AValue: Integer)`

Visibility: default

Description: `IntegerValue` is called when a JSON integer is encountered during parsing. The method receives the integer value that was parsed from the JSON data. Implementers should handle the integer value according to their processing requirements.

Errors: Implementation-specific error handling.

46.8.10 IJSONConsumer.StringValue

Synopsis: Processes a JSON string value

Declaration: `procedure StringValue(const AValue: TJSONStringType)`

Visibility: default

Description: `StringValue` is called when a JSON string is encountered during parsing. The method receives the string value that was parsed from the JSON data. Implementers should handle the string value according to their processing requirements.

Errors: Implementation-specific error handling.

46.8.11 IJSONConsumer.KeyName

Synopsis: Processes a JSON object key name

Declaration: `procedure KeyName(const AKey: TJSONStringType)`

Visibility: default

Description: `KeyName` is called when a JSON object key name is encountered during parsing. The method receives the key name that was parsed from the JSON object data. Implementers should handle the key name according to their processing requirements.

Errors: Implementation-specific error handling.

46.8.12 IJSONConsumer.StartObject

Synopsis: Processes the start of a JSON object

Declaration: `procedure StartObject`

Visibility: default

Description: `StartObject` is called when the beginning of a JSON object is encountered during parsing. This corresponds to opening curly braces in the JSON data. Implementers should handle the start of the object according to their processing requirements.

Errors: Implementation-specific error handling.

See also: EndObject (1335)

46.8.13 IJSONConsumer.EndObject

Synopsis: Processes the end of a JSON object

Declaration: `procedure EndObject`

Visibility: default

Description: `EndObject` is called when the end of a JSON object is encountered during parsing. This corresponds to closing curly braces in the JSON data. Implementers should handle the end of the object according to their processing requirements.

Errors: Implementation-specific error handling.

See also: StartObject (1335)

46.8.14 IJSONConsumer.StartArray

Synopsis: Processes the start of a JSON array

Declaration: `procedure StartArray`

Visibility: default

Description: `StartArray` is called when the beginning of a JSON array is encountered during parsing. This corresponds to opening square brackets in the JSON data. Implementers should handle the start of the array according to their processing requirements.

Errors: Implementation-specific error handling.

See also: EndArray (1336)

46.8.15 IJSONConsumer.EndArray

Synopsis: Processes the end of a JSON array

Declaration: `procedure EndArray`

Visibility: default

Description: `EndArray` is called when the end of a JSON array is encountered during parsing. This corresponds to closing square brackets in the JSON data. Implementers should handle the end of the array according to their processing requirements.

Errors: Implementation-specific error handling.

See also: StartArray (1335)

46.9 TBaseJSONReader

46.9.1 Description

`TBaseJSONReader` is the abstract base class for all JSON reading functionality. It provides the core parsing logic and virtual methods that must be implemented by derived classes. The class uses TJSONScanner (1323) for tokenization and handles the parsing of JSON objects, arrays, and primitive values. Derived classes must implement abstract methods to handle different JSON element types encountered during parsing.

See also: TJSONEventReader (1338), TJSONConsumerReader (1337), TJSONScanner (1323)

46.9.2 Method overview

Page	Method	Description
1336	Create	Creates a new JSON reader with specified source and options
1337	Destroy	Destroys the JSON reader and frees resources

46.9.3 Property overview

Page	Properties	Access	Description
1337	Options	rw	JSON parsing options that control reader behavior

46.9.4 TBaseJSONReader.Create

Synopsis: Creates a new JSON reader with specified source and options

Declaration:
```
constructor Create(Source: TStream; AUseUTF8: Boolean);  Overload
constructor Create(const Source: RawByteString; AUseUTF8: Boolean)
                   ;  Overload
constructor Create(Source: TStream; AOptions: TJSONOptions);  Overload
constructor Create(const Source: RawByteString; AOptions: TJSONOptions)
                   ;  Overload
constructor Create(const Source: UnicodeString; AOptions: TJSONOptions)
                   ;  Overload
```

Visibility: public

Description: Creates a new `TBaseJSONReader` instance with the specified JSON source and parsing options. The constructor initializes the internal TJSONScanner (1323) with the provided source and options. Multiple overloaded versions are available for different source types and option configurations.

Errors: May raise exceptions if the source cannot be accessed or is invalid.

See also: Destroy (1337), Options (1337)

46.9.5 TBaseJSONReader.Destroy

Synopsis: Destroys the JSON reader and frees resources

Declaration: `destructor Destroy;  Override`

Visibility: public

Description: Destroys the `TBaseJSONReader` instance and frees all associated resources. This includes releasing the internal TJSONScanner (1323) and any allocated memory.

See also: Create (1336)

46.9.6 TBaseJSONReader.Options

Synopsis: JSON parsing options that control reader behavior

Declaration: `Property Options : TJSONOptions`

Visibility: public

Access: Read,Write

Description: The `Options` property controls various aspects of JSON parsing behavior. This includes UTF-8 handling, strict mode parsing, and other parsing preferences. Changes to options affect how the underlying scanner processes JSON tokens.

See also: TJSONOptions (1323), TJSONScanner (1323)

46.10 TJSONConsumerReader

46.10.1 Description

`TJSONConsumerReader` provides JSON reading functionality using the IJSONConsumer (1332) interface. It inherits from TBaseJSONReader (1336) and delegates JSON element processing to an assigned consumer object. This approach allows for flexible JSON processing by implementing the IJSONConsumer (1332) interface rather than creating event handlers. The consumer pattern provides an alternative to the event-driven approach used by TJSONEventReader (1338).

See also: TBaseJSONReader (1336), IJSONConsumer (1332), TJSONEventReader (1338)

46.10.2 Method overview

Page	Method	Description
1338	Execute	Starts the JSON parsing process using the assigned consumer

46.10.3 Property overview

Page	Properties	Access	Description
1338	Consumer	rw	JSON consumer object that processes parsed elements

46.10.4 TJSONConsumerReader.Execute

Synopsis: Starts the JSON parsing process using the assigned consumer

Declaration: `procedure Execute`

Visibility: public

Description: `Execute` starts the JSON parsing process. As JSON elements are encountered during parsing, the corresponding methods in the assigned IJSONConsumer (1332) are called if a consumer is assigned. The method processes the entire JSON input until completion or until an error is encountered.

Errors: Raises EJSONParser (1332) when invalid JSON syntax is encountered.

See also: TBaseJSONReader.DoExecute (1336), Consumer (1338)

46.10.5 TJSONConsumerReader.Consumer

Synopsis: JSON consumer object that processes parsed elements

Declaration: `Property Consumer : IJSONConsumer`

Visibility: public

Access: Read,Write

Description: `Consumer` specifies the IJSONConsumer (1332) object that will process JSON elements as they are parsed. When assigned, the consumer's methods are called for each JSON element type encountered during parsing. If no consumer is assigned, the parsing will proceed but no processing will occur.

See also: IJSONConsumer (1332)

46.11 TJSONEventReader

46.11.1 Description

`TJSONEventReader` provides event-based JSON reading functionality. It inherits from TBaseJSONReader (1336) and fires specific events when different JSON elements are encountered during parsing. This allows applications to process JSON data using an event-driven approach similar to SAX parsing for XML. Each JSON element type has a corresponding event property that can be assigned to handle the parsed data.

See also: TBaseJSONReader (1336), TJSONConsumerReader (1337)

46.11.2 Method overview

Page	Method	Description
1339	Execute	Starts the JSON parsing process and fires events

46.11.3 Property overview

Page	Properties	Access	Description
1339	OnBooleanValue	rw	Event fired when a JSON boolean value is encountered
1343	OnEndArray	rw	Event fired when a JSON array ends
1342	OnEndObject	rw	Event fired when a JSON object ends
1340	OnFloatValue	rw	Event fired when a JSON floating-point value is encountered
1341	OnInt64Value	rw	Event fired when a JSON 64-bit integer value is encountered
1340	OnIntegerValue	rw	Event fired when a JSON integer value is encountered
1341	OnKeyName	rw	Event fired when a JSON object key name is encountered
1339	OnNullValue	rw	Event fired when a JSON null value is encountered
1340	OnNumberValue	rw	Event fired when a JSON number value is encountered
1341	OnQWordValue	rw	Event fired when a JSON quadword value is encountered
1342	OnStartArray	rw	Event fired when a JSON array begins
1342	OnStartObject	rw	Event fired when a JSON object begins
1341	OnStringValue	rw	Event fired when a JSON string value is encountered

46.11.4 TJSONEventReader.Execute

Synopsis: Starts the JSON parsing process and fires events

Declaration: `procedure Execute`

Visibility: public

Description: `Execute` starts the JSON parsing process. As JSON elements are encountered during parsing, the corresponding event handlers are called if they are assigned. The method processes the entire JSON input until completion or until an error is encountered.

Errors: Raises EJSONParser (1332) when invalid JSON syntax is encountered.

See also: TBaseJSONReader.DoExecute (1336)

46.11.5 TJSONEventReader.OnNullValue

Synopsis: Event fired when a JSON null value is encountered

Declaration: `Property OnNullValue : TNotifyEvent`

Visibility: public

Access: Read,Write

Description: `OnNullValue` is fired when a JSON null value is encountered during parsing. The event handler receives the sender object as parameter. This event corresponds to JSON null values in the input data.

See also: TNotifyEvent (1323)

46.11.6 TJSONEventReader.OnBooleanValue

Synopsis: Event fired when a JSON boolean value is encountered

Declaration: `Property OnBooleanValue : TOnJSONBoolean`

Visibility: public

Access: Read,Write

Description: `OnBooleanValue` is fired when a JSON boolean value is encountered during parsing. The event handler receives the sender object and the boolean value as parameters. This event corresponds to JSON `True` and `False` values in the input data.

See also: TOnJSONBoolean (1331)

46.11.7 TJSONEventReader.OnNumberValue

Synopsis: Event fired when a JSON number value is encountered

Declaration: `Property OnNumberValue : TOnJSONString`

Visibility: public

Access: Read,Write

Description: `OnNumberValue` is fired when any JSON number value is encountered during parsing. The event handler receives the sender object and the number as a string representation. This event is fired for all numeric values before they are converted to specific numeric types.

See also: TOnJSONString (1332)

46.11.8 TJSONEventReader.OnFloatValue

Synopsis: Event fired when a JSON floating-point value is encountered

Declaration: `Property OnFloatValue : TOnJSONFloat`

Visibility: public

Access: Read,Write

Description: `OnFloatValue` is fired when a JSON floating-point number is encountered during parsing. The event handler receives the sender object and the floating-point value as parameters. This event is fired for JSON numbers that contain decimal points or exponential notation.

See also: TOnJSONFloat (1331)

46.11.9 TJSONEventReader.OnIntegerValue

Synopsis: Event fired when a JSON integer value is encountered

Declaration: `Property OnIntegerValue : TOnJSONInteger`

Visibility: public

Access: Read,Write

Description: `OnIntegerValue` is fired when a JSON integer value that fits in a standard integer is encountered during parsing. The event handler receives the sender object and the integer value as parameters. This event is fired for JSON numbers that can be represented as 32-bit signed integers.

See also: TOnJSONInteger (1331)

46.11.10 TJSONEventReader.OnInt64Value

Synopsis: Event fired when a JSON 64-bit integer value is encountered

Declaration: `Property OnInt64Value : TOnJSONInt64`

Visibility: public

Access: Read,Write

Description: `OnInt64Value` is fired when a JSON integer value that requires 64-bit representation is encountered during parsing. The event handler receives the sender object and the 64-bit integer value as parameters. This event is fired for JSON numbers that exceed the range of 32-bit integers but fit in 64-bit signed integers.

See also: TOnJSONInt64 (1331)

46.11.11 TJSONEventReader.OnQWordValue

Synopsis: Event fired when a JSON quadword value is encountered

Declaration: `Property OnQWordValue : TOnJSONQWord`

Visibility: public

Access: Read,Write

Description: `OnQWordValue` is fired when a JSON integer value that requires unsigned 64-bit representation is encountered during parsing. The event handler receives the sender object and the quadword value as parameters. This event is fired for JSON numbers that exceed the range of signed 64-bit integers.

See also: TOnJSONQWord (1332)

46.11.12 TJSONEventReader.OnStringValue

Synopsis: Event fired when a JSON string value is encountered

Declaration: `Property OnStringValue : TOnJSONString`

Visibility: public

Access: Read,Write

Description: `OnStringValue` is fired when a JSON string value is encountered during parsing. The event handler receives the sender object and the string value as parameters. This event corresponds to JSON string literals enclosed in double quotes.

See also: TOnJSONString (1332)

46.11.13 TJSONEventReader.OnKeyName

Synopsis: Event fired when a JSON object key name is encountered

Declaration: `Property OnKeyName : TOnJSONKey`

Visibility: public

Access: Read,Write

Description: `OnKeyName` is fired when a JSON object key name is encountered during parsing. The event handler receives the sender object and the key name as parameters. This event is fired for the key portion of key-value pairs in JSON objects.

See also: TOnJSONKey (1331)

46.11.14 TJSONEventReader.OnStartObject

Synopsis: Event fired when a JSON object begins

Declaration: `Property OnStartObject : TNotifyEvent`

Visibility: public

Access: Read,Write

Description: `OnStartObject` is fired when the start of a JSON object is encountered during parsing. The event handler receives the sender object as parameter. This event corresponds to opening curly braces in JSON data.

See also: OnEndObject (1342), TNotifyEvent (1323)

46.11.15 TJSONEventReader.OnEndObject

Synopsis: Event fired when a JSON object ends

Declaration: `Property OnEndObject : TNotifyEvent`

Visibility: public

Access: Read,Write

Description: `OnEndObject` is fired when the end of a JSON object is encountered during parsing. The event handler receives the sender object as parameter. This event corresponds to closing curly braces in JSON data.

See also: OnStartObject (1342), TNotifyEvent (1323)

46.11.16 TJSONEventReader.OnStartArray

Synopsis: Event fired when a JSON array begins

Declaration: `Property OnStartArray : TNotifyEvent`

Visibility: public

Access: Read,Write

Description: `OnStartArray` is fired when the start of a JSON array is encountered during parsing. The event handler receives the sender object as parameter. This event corresponds to opening square brackets in JSON data.

See also: OnEndArray (1343), TNotifyEvent (1323)

46.11.17 TJSONEventReader.OnEndArray

Synopsis: Event fired when a JSON array ends

Declaration: `Property OnEndArray : TNotifyEvent`

Visibility: public

Access: Read,Write

Description: `OnEndArray` is fired when the end of a JSON array is encountered during parsing. The event handler receives the sender object as parameter. This event corresponds to closing square brackets in JSON data.

See also: OnStartArray (1342), TNotifyEvent (1323)

Chapter 47

Reference for unit 'jsonscanner'

47.1 Used units

Table 47.1: Used units by unit 'jsonscanner'

Name	Page
Classes	??
System	??
sysutils	??

47.2 Overview

The `jsonscanner` unit provides a tokenizer for JSON data. It breaks JSON text into discrete tokens for subsequent parsing.

47.3 Constants, types and variables

47.3.1 Resource strings

```
SErrInvalidCharacter =
  'Invalid character at line %d, pos %d: ''%s'''
```

Error message format string for invalid characters encountered during scanning. Contains placeholders for line number, column position, and the invalid character.

```
SErrOpenString = 'string exceeds end of line %d'
```

Error message format string for unclosed strings that exceed the end of a line. Contains a placeholder for the line number where the string begins.

```
SUnterminatedComment =
  'Unterminated comment at line %d, pos %d: ''%s'''
```

Error message format string for unterminated comments encountered during scanning. Contains placeholders for line number and column position where the unterminated comment begins.

47.3.2 Constants

```
DefaultOptions = [joUTF8]
```

Default set of options for the JSON scanner. Contains the joUTF8 (1345) option by default.

```
TokenInfos : Array[TJSONToken] of string = ('EOF', 'Whitespace', 'String'
  , 'Number', 'True', 'False', 'Null', ',', ':', '{', '}', '[', ']'
  , 'identifier', 'comment', '')
```

Array of string descriptions for each TJSONToken (1346) type. Provides human-readable names for token types, useful for debugging and error reporting. Indexed by token type enumeration values.

47.3.3 Types

```
TJSONOption = (joUTF8,joStrict,joComments,joIgnoreTrailingComma,
  joIgnoreDuplicates,joBOMCheck)
```

Table 47.2: Enumeration values for type TJSONOption

Value	Explanation
joBOMCheck	Check byte order mark option
joComments	Allow comments option
joIgnoreDuplicates	Ignore duplicate keys option
joIgnoreTrailingComma	Ignore trailing comma option
joStrict	Strict JSON parsing option
joUTF8	UTF8 string encoding option

Enumeration of options that control the behavior of the JSON scanner. These options can be combined in a TJSONOptions (1345) set.

joUTF8 Return strings with UTF8 codepage

joStrict Use strict JSON parsing (require quotes around object member names)

joComments Allow JavaScript-style comments in JSON

joIgnoreTrailingComma Ignore trailing commas in arrays and objects

joIgnoreDuplicates Do not attempt to add duplicate object members

joBOMCheck Check for byte order mark at beginning of stream

joSingle Read only a single JSON value from the stream

```
TJSONOptions = Set of TJSONOption
```

Set type that combines multiple TJSONOption (1345) values. Used to configure the behavior of the JSON scanner.

```
TJSONToken = (tkEOF,tkWhitespace,tkString,tkNumber,tkTrue,tkFalse
  ,
              tkNull,tkComma,tkColon,tkCurlyBraceOpen,tkCurlyBraceClose
  ,
              tkSquaredBraceOpen,tkSquaredBraceClose,tkIdentifier
  ,
              tkComment,tkUnknown)
```

Table 47.3: Enumeration values for type TJSONToken

Value	Explanation
tkColon	Colon separator token
tkComma	Comma separator token
tkComment	Comment token
tkCurlyBraceClose	Closing curly brace token
tkCurlyBraceOpen	Opening curly brace token
tkEOF	End of file token
tkFalse	Boolean false token
tkIdentifier	Identifier token
tkNull	Null value token
tkNumber	Numeric value token
tkSquaredBraceClose	Closing square bracket token
tkSquaredBraceOpen	Opening square bracket token
tkString	String literal token
tkTrue	Boolean true token
tkUnknown	Unknown token
tkWhitespace	Whitespace token

Enumeration of all possible token types that can be encountered during JSON scanning. Each token represents a distinct syntactic element in JSON syntax.

tkEOF End of input stream reached

tkWhitespace Whitespace characters (spaces, tabs)

tkString String literal enclosed in quotes

tkNumber Numeric value (integer or floating-point)

tkTrue Boolean `True` literal

tkFalse Boolean `False` literal

tkNull Null literal value

tkComma Comma separator character

tkColon Colon separator character

tkCurlyBraceOpen Opening curly brace for objects

tkCurlyBraceClose Closing curly brace for objects

tkSquaredBraceOpen Opening square bracket for arrays

tkSquaredBraceClose Closing square bracket for arrays

tkIdentifier Identifier (in non-strict mode)

tkComment Comment content (when comments are enabled)

tkUnknown Unrecognized token

47.4 EScannerError

47.4.1 Description

Exception class raised when errors occur during JSON tokenization. Descends from `EParserError` to provide specific error handling for scanner operations.

See also: SErrInvalidCharacter (1344), SUnterminatedComment (1344), SErrOpenString (1344)

47.5 TJSONScanner

47.5.1 Description

Lexical scanner class that tokenizes JSON text input. Breaks JSON source into discrete tokens for subsequent parsing by higher-level parsers. Supports various scanning options through TJSONOptions (1345).

See also: TJSONToken (1346), TJSONOptions (1345), EScannerError (1347)

47.5.2 Method overview

Page	Method	Description
1347	Create	Create JSON scanner instance
1348	FetchToken	Fetch next token from input

47.5.3 Property overview

Page	Properties	Access	Description
1349	CurColumn	r	Current column number
1348	CurLine	r	Current line content
1348	CurRow	r	Current row number
1349	CurToken	r	Current token type
1349	CurTokenString	r	Current token string value
1350	Options	rw	Scanner options set
1349	Strict	rw	Strict JSON parsing mode
1350	UseUTF8	rw	UTF-8 encoding mode

47.5.4 TJSONScanner.Create

Synopsis: Create JSON scanner instance

Declaration:
```
constructor Create(Source: TStream; AUseUTF8: Boolean);  Overload
constructor Create(Source: TStream; AOptions: TJSONOptions);  Overload
constructor Create(const aSource: RawByteString; AUseUTF8: Boolean)
```

```
                    ;  Overload
constructor Create(const aSource: RawByteString; AOptions: TJSONOptions)
                    ;  Overload
```

Visibility: public

Description: Creates a new JSON scanner instance. Can be initialized with either a `TStream` or a raw byte string source. Accepts optional scanning options to control parsing behavior.

Errors: May raise exceptions during source loading from streams.

See also: TJSONOptions (1345), DefaultOptions (1345)

47.5.5 TJSONScanner.FetchToken

Synopsis: Fetch next token from input

Declaration: `function FetchToken : TJSONToken`

Visibility: public

Description: Reads and returns the next token from the JSON input stream. Advances the scanner position and updates current token information. The token type and associated string value are available through properties after calling this method.

Errors: Raises EScannerError (1347) when encountering invalid JSON syntax.

See also: TJSONToken (1346), CurToken (1349), CurTokenString (1349)

47.5.6 TJSONScanner.CurLine

Synopsis: Current line content

Declaration: `Property CurLine : Ansistring`

Visibility: public

Access: Read

Description: Returns the content of the current line being scanned. Useful for error reporting and debugging.

See also: CurRow (1348), CurColumn (1349)

47.5.7 TJSONScanner.CurRow

Synopsis: Current row number

Declaration: `Property CurRow : Integer`

Visibility: public

Access: Read

Description: Returns the current line number in the input stream. Line numbers start from 1.

See also: CurLine (1348), CurColumn (1349)

47.5.8 TJSONScanner.CurColumn

Synopsis: Current column number

Declaration: `Property CurColumn : Integer`

Visibility: public

Access: Read

Description: Returns the current column position within the current line. Column numbers start from 0.

See also: CurRow (1348), CurLine (1348)

47.5.9 TJSONScanner.CurToken

Synopsis: Current token type

Declaration: `Property CurToken : TJSONToken`

Visibility: public

Access: Read

Description: Returns the type of the current token. Updated after each call to FetchToken (1348).

See also: TJSONToken (1346), FetchToken (1348), CurTokenString (1349)

47.5.10 TJSONScanner.CurTokenString

Synopsis: Current token string value

Declaration: `Property CurTokenString : Ansistring`

Visibility: public

Access: Read

Description: Returns the string representation of the current token. For string tokens, contains the unescaped string content. For other tokens, contains the literal text representation.

See also: CurToken (1349), FetchToken (1348)

47.5.11 TJSONScanner.Strict

Synopsis: Strict JSON parsing mode

Declaration: `Property Strict : Boolean; deprecated;`

Visibility: public

Access: Read,Write

Description: Controls whether strict JSON parsing is enabled. When `True`, requires quotes around object member names and disallows single quotes. This property is deprecated; use Options (1350) instead.

See also: joStrict (1345), Options (1350)

47.5.12 TJSONScanner.UseUTF8

Synopsis: UTF-8 encoding mode

Declaration: `Property UseUTF8 : Boolean; deprecated;`

Visibility: public

Access: Read,Write

Description: Controls whether strings are converted to UTF-8 encoding. When `True`, string tokens use UTF-8 codepage instead of system codepage. This property is deprecated; use Options (1350) instead.

See also: joUTF8 (1345), Options (1350)

47.5.13 TJSONScanner.Options

Synopsis: Scanner options set

Declaration: `Property Options : TJSONOptions`

Visibility: public

Access: Read,Write

Description: Set of options that control the scanner behavior. Allows combining multiple TJSONOption (1345) values to customize parsing.

See also: TJSONOptions (1345), TJSONOption (1345), DefaultOptions (1345)

Chapter 48

Reference for unit 'libtar'

48.1 Used units

Table 48.1: Used units by unit 'libtar'

Name	Page
BaseUnix	??
Classes	??
System	??
sysutils	??
Unix	??
UnixType	??

48.2 Overview

The libtar units provides 2 classes to read and write .tar archives: TTarArchive (1355) class can be used to read a tar file, and the TTarWriter (1357) class can be used to write a tar file. The unit was implemented originally by Stefan Heymann.

48.3 Constants, types and variables

48.3.1 Constants

```
ALL_PERMISSIONS = [tpReadByOwner, tpWriteByOwner, tpExecuteByOwner
  , tpReadByGroup, tpWriteByGroup, tpExecuteByGroup, tpReadByOther,
  tpWriteByOther, tpExecuteByOther]
```

ALL_PERMISSIONS is a set constant containing all possible permissions (read/write/execute, for all groups of users) for an archive entry.

```
EXECUTE_PERMISSIONS = [tpExecuteByOwner, tpExecuteByGroup, tpExecuteByOther
  ]
```

WRITE_PERMISSIONS is a set constant containing all possible execute permissions set for an archive entry.

```
FILETYPE_NAME : Array[TFileType] of string = ('Regular', 'Link', 'Symbolic Link'
  , 'Char File', 'Block File', 'Directory', 'FIFO File', 'Contiguous'
  , 'Dir Dump', 'Multivol', 'Volume Header')
```

FILETYPE_NAME can be used to get a textual description for each of the possible entry file types.

```
READ_PERMISSIONS = [tpReadByOwner, tpReadByGroup, tpReadByOther]
```

READ_PERMISSIONS is a set constant containing all possible read permissions set for an archive entry.

```
WRITE_PERMISSIONS = [tpWriteByOwner, tpWriteByGroup, tpWriteByOther
  ]
```

WRITE_PERMISSIONS is a set constant containing all possible write permissions set for an archive entry.

48.3.2 Types

```
TFileType = (ftNormal,ftLink,ftSymbolicLink,ftCharacter,ftBlock,
  ftDirectory,ftFifo,ftContiguous,ftDumpDir,ftMultiVolume
  ,
             ftVolumeHeader)
```

Table 48.2: Enumeration values for type TFileType

Value	Explanation
ftBlock	Block device file.
ftCharacter	Character device file.
ftContiguous	Contiguous file.
ftDirectory	Directory.
ftDumpDir	List of files.
ftFifo	FIFO file.
ftLink	Hard link.
ftMultiVolume	Multi-volume file part.
ftNormal	Normal file.
ftSymbolicLink	Symbolic link.
ftVolumeHeader	Volume header, can appear only as first entry in the archive.

TFileType describes the file type of a file in the archive. It is used in the FileType field of the TTarDirRec (1353) record.

```
TTarDirRec = record
public
  Name : AnsiString;
  Size : Int64;
  DateTime : TDateTime;
  Permissions : TTarPermissions;
  FileType
  : TFileType;
```

```
  LinkName : AnsiString;
  UID : Integer;
  GID : Integer
  ;
  UserName : AnsiString;
  GroupName : AnsiString;
  ChecksumOK
  : Boolean;
  Mode : TTarModes;
  Magic : AnsiString;
  MajorDevNo
  : Integer;
  MinorDevNo : Integer;
  FilePos : Int64;
end
```

`TTarDirRec` describes an entry in the tar archive. It is similar to a directory entry as in TSearchRec (**??**), and is returned by the TTarArchive.FindNext (1356) call.

```
TTarMode = (tmSetUid,tmSetGid,tmSaveText)
```

Table 48.3: Enumeration values for type TTarMode

Value	Explanation
tmSaveText	Bit $200 is set.
tmSetGid	File has SetGID bit set.
tmSetUid	File has SetUID bit set.

`TTarMode` describes extra file modes. It is used in the `Mode` field of the TTarDirRec (1353) record.

```
TTarModes = Set of TTarMode
```

`TTarModes` denotes the full set of permission bits for the file in the field `Mode` field of the TTarDirRec (1353) record.

```
TTarPermission = (tpReadByOwner,tpWriteByOwner,tpExecuteByOwner,
  tpReadByGroup,tpWriteByGroup,tpExecuteByGroup,
  tpReadByOther,tpWriteByOther,tpExecuteByOther)
```

Table 48.4: Enumeration values for type TTarPermission

Value	Explanation
tpExecuteByGroup	Group can execute the file.
tpExecuteByOther	Other people can execute the file.
tpExecuteByOwner	Owner can execute the file.
tpReadByGroup	Group can read the file.
tpReadByOther	Other people can read the file.
tpReadByOwner	Owner can read the file.
tpWriteByGroup	Group can write the file.
tpWriteByOther	Other people can write the file.
tpWriteByOwner	Owner can write the file.

TTarPermission denotes part of a files permission as it stored in the .tar archive. Each of these enumerated constants correspond with one of the permission bits from a UNIX file permission.

```
TTarPermissions = Set of TTarPermission
```

TTarPermissions describes the complete set of permissions that a file has. It is used in the Permissions field of the TTarDirRec (1353) record.

48.4 Procedures and functions

48.4.1 ClearDirRec

Synopsis: Initialize tar archive entry.

Declaration:
```
procedure ClearDirRec(var DirRec: TTarDirRec)
```

Visibility: default

Description: ClearDirRec clears the DirRec entry, it basically zeroes out all fields.

See also: TTarDirRec (1353)

48.4.2 ConvertFilename

Synopsis: Convert filename to archive format.

Declaration:
```
function ConvertFilename(const Filename: AnsiString) : AnsiString
```

Visibility: default

Description: ConvertFileName converts the file name FileName to a format allowed by the tar archive. Basically, it converts directory specifiers to forward slashes.

48.4.3 FileTimeGMT

Synopsis: Extract filetime.

Declaration:
```
function FileTimeGMT(const FileName: AnsiString) : TDateTime;  Overload
function FileTimeGMT(SearchRec: TSearchRec) : TDateTime;  Overload
```

Visibility: default

Description: FileTimeGMT returns the timestamp of a filename (FileName must exist) or a search rec (TSearchRec) to a GMT representation that can be used in a tar entry.

See also: TTarDirRec (1353)

48.4.4 PermissionString

Synopsis: Convert a set of permissions to a string.

Declaration:
```
function PermissionString(Permissions: TTarPermissions) : string
```

Visibility: default

Description: PermissionString can be used to convert a set of Permissions to a string in the same format as used by the UNIX 'ls' command.

See also: TTarPermissions (1354)

48.5 TTarArchive

48.5.1 Description

`TTarArchive` is the class used to read and examine .tar archives. It can be constructed from a stream or from a filename. Creating an instance will not perform any operation on the stream yet.

See also: TTarWriter (1357), FindNext (1356)

48.5.2 Method overview

Page	Method	Description
1355	Create	Create a new instance of the archive.
1355	Destroy	Destroy `TTarArchive` instance.
1356	FindNext	Find next archive entry.
1357	GetFilePos	Return current archive position.
1356	ReadFile	Read a file from the archive.
1356	Reset	Reset archive.
1357	SetFilePos	Set position in archive.

48.5.3 TTarArchive.Create

Synopsis: Create a new instance of the archive.

Declaration:
```
constructor Create(Stream: TStream);  Overload
constructor Create(const Filename: AnsiString; FileMode: Word)
                   ;  Overload
```

Visibility: public

Description: `Create` can be used to create a new instance of `TTarArchive` using either a `Stream`TStream (**??**) descendent or using a name of a file to open: `FileName`. In case of the filename, an open mode can be specified.

Errors: In case a filename is specified and the file cannot be opened, an exception will occur.

See also: FindNext (1356)

48.5.4 TTarArchive.Destroy

Synopsis: Destroy `TTarArchive` instance.

Declaration:
```
destructor Destroy;  Override
```

Visibility: public

Description: `Destroy` closes the archive stream (if it created a stream) and cleans up the `TTarArchive` instance.

See also: TTarArchive.Create (1355)

48.5.5 TTarArchive.Reset

Synopsis: Reset archive.

Declaration:
```
procedure Reset
```

Visibility: public

Description: `Reset` sets the archive file position on the beginning of the archive.

See also: TTarArchive.Create (1355)

48.5.6 TTarArchive.FindNext

Synopsis: Find next archive entry.

Declaration:
```
function FindNext(var DirRec: TTarDirRec) : Boolean
```

Visibility: public

Description: `FindNext` positions the file pointer on the next archive entry, and returns all information about the entry in `DirRec`. It returns `True` if the operation was successful, or `False` if not (for instance, when the end of the archive was reached).

Errors: In case there are no more entries, `False` is returned.

See also: TTarArchive.ReadFile (1356)

48.5.7 TTarArchive.ReadFile

Synopsis: Read a file from the archive.

Declaration:
```
procedure ReadFile(Buffer: POINTER);  Overload
procedure ReadFile(Stream: TStream);  Overload
procedure ReadFile(Filename: string);  Overload
function ReadFile : string;  Overload
```

Visibility: public

Description: `ReadFile` can be used to read the current file in the archive. It can be called after the archive was successfully positioned on an entry in the archive. The file can be read in various ways:

- directly in a memory `buffer`. No checks are performed to see whether the buffer points to enough memory.
- It can be copied to a `Stream`.
- It can be copied to a file with name `FileName`.
- The file content can be copied to a string

Errors: An exception may occur if the buffer is not large enough, or when the file specified in `filename` cannot be opened.

48.5.8 TTarArchive.GetFilePos

Synopsis: Return current archive position.

Declaration: `procedure GetFilePos(var Current: Int64; var Size: Int64)`

Visibility: public

Description: `GetFilePos` returns the position in the tar archive in `Current` and the complete archive size in `Size`.

See also: TTarArchive.SetFilePos (1357), TTarArchive.Reset (1356)

48.5.9 TTarArchive.SetFilePos

Synopsis: Set position in archive.

Declaration: `procedure SetFilePos(NewPos: Int64)`

Visibility: public

Description: `SetFilePos` can be used to set the absolute position in the tar archive.

See also: TTarArchive.Reset (1356), TTarArchive.GetFilePos (1357)

48.6 TTarWriter

48.6.1 Description

`TTarWriter` can be used to create .tar archives. It can be created using a filename, in which case the archive will be written to the filename, or it can be created using a stream, in which case the archive will be written to the stream - for instance a compression stream.

See also: TTarArchive (1355)

48.6.2 Method overview

Page	Method	Description
1359	AddDir	Add directory to archive.
1358	AddFile	Add a file to the archive.
1360	AddLink	Add hard link to archive.
1359	AddStream	Add stream contents to archive.
1359	AddString	Add string as file data.
1360	AddSymbolicLink	Add a symbolic link to the archive.
1360	AddVolumeHeader	Add volume header entry.
1358	Create	Create a new archive.
1358	Destroy	Close archive and clean up `TTarWriter`.
1360	Finalize	Finalize the archive.

48.6.3 Property overview

Page	Properties	Access	Description
1361	GID	rw	Archive entry group ID.
1362	GroupName	rw	Archive entry group name.
1362	Magic	rw	Archive entry Magic constant.
1362	Mode	rw	Archive entry mode.
1361	Permissions	rw	Archive entry permissions.
1361	UID	rw	Archive entry user ID.
1361	UserName	rw	Archive entry user name.

48.6.4 TTarWriter.Create

Synopsis: Create a new archive.

Declaration:
```
constructor Create(TargetStream: TStream);  Overload
constructor Create(const TargetFilename: AnsiString; Mode: Integer)
                  ;  Overload
```

Visibility: public

Description: `Create` creates a new `TTarWriter` instance. This will start a new .tar archive. The archive will be written to the `TargetStream` stream or to a file with name `TargetFileName`, which will be opened with filemode `Mode`.

Errors: In case `TargetFileName` cannot be opened, an exception will be raised.

See also: TTarWriter.Destroy (1358)

48.6.5 TTarWriter.Destroy

Synopsis: Close archive and clean up `TTarWriter`.

Declaration:
```
destructor Destroy;  Override
```

Visibility: public

Description: `Destroy` will close the archive (i.e. it writes the end-of-archive marker, if it was not yet written), and then frees the `TTarWriter` instance.

See also: TTarWriter.Finalize (1360)

48.6.6 TTarWriter.AddFile

Synopsis: Add a file to the archive.

Declaration:
```
procedure AddFile(const Filename: AnsiString; TarFilename: AnsiString)
```

Visibility: public

Description: `AddFile` adds a file to the archive: the contents is read from `FileName`. Optionally, an alternative filename can be specified in `TarFileName`. This name should contain only forward slash path separators. If it is not specified, the name will be computed from `FileName`.

The archive entry is written with the current owner data and permissions.

Errors: If `FileName` cannot be opened, an exception will be raised.

See also: TTarWriter.AddStream (1359), TTarWriter.AddString (1359), TTarWriter.AddLink (1360), TTarWriter.AddSymbolicLink (1360), TTarWriter.AddDir (1359), TTarWriter.AddVolumeHeader (1360)

48.6.7 TTarWriter.AddStream

Synopsis: Add stream contents to archive.

Declaration:
```
procedure AddStream(Stream: TStream; TarFilename: AnsiString;
                    FileDateGmt: TDateTime)
```

Visibility: public

Description: `AddStream` will add the contents of `Stream` to the archive. The Stream will not be reset: only the contents of the stream from the current position will be written to the archive. The entry will be written with file name `TarFileName`. This name should contain only forward slash path separators. The entry will be written with timestamp `FileDateGmt`.

The archive entry is written with the current owner data and permissions.

See also: TTarWriter.AddFile (1358), TTarWriter.AddString (1359), TTarWriter.AddLink (1360), TTarWriter.AddSymbolicLink (1360), TTarWriter.AddDir (1359), TTarWriter.AddVolumeHeader (1360)

48.6.8 TTarWriter.AddString

Synopsis: Add string as file data.

Declaration:
```
procedure AddString(Contents: Ansistring; TarFilename: AnsiString;
                    FileDateGmt: TDateTime)
```

Visibility: public

Description: `AddString` adds the string `Contents` as the data of an entry with file name `TarFileName`. This name should contain only forward slash path separators. The entry will be written with timestamp `FileDateGmt`.

The archive entry is written with the current owner data and permissions.

See also: TTarWriter.AddFile (1358), TTarWriter.AddStream (1359), TTarWriter.AddLink (1360), TTarWriter.AddSymbolicLink (1360), TTarWriter.AddDir (1359), TTarWriter.AddVolumeHeader (1360)

48.6.9 TTarWriter.AddDir

Synopsis: Add directory to archive.

Declaration:
```
procedure AddDir(Dirname: AnsiString; DateGmt: TDateTime;
                 MaxDirSize: Int64)
```

Visibility: public

Description: `AddDir` adds a directory entry to the archive. The entry is written with name `DirName`, maximum directory size `MaxDirSize` (0 means unlimited) and timestamp `DateGmt`.

Note that this call only adds an entry for a directory to the archive: if `DirName` is an existing directory, it does not write all files in the directory to the archive.

The directory entry is written with the current owner data and permissions.

See also: TTarWriter.AddFile (1358), TTarWriter.AddStream (1359), TTarWriter.AddLink (1360), TTarWriter.AddSymbolicLink (1360), TTarWriter.AddString (1359), TTarWriter.AddVolumeHeader (1360)

48.6.10 TTarWriter.AddSymbolicLink

Synopsis: Add a symbolic link to the archive.

Declaration:
```
procedure AddSymbolicLink(Filename: AnsiString; Linkname: AnsiString;
                          DateGmt: TDateTime)
```

Visibility: public

Description: `AddSymbolicLink` adds a symbolic link entry to the archive, with name `FileName`, pointing to `LinkName`. The entry is written with timestamp `DateGmt`.

The link entry is written with the current owner data and permissions.

See also: TTarWriter.AddFile (1358), TTarWriter.AddStream (1359), TTarWriter.AddLink (1360), TTarWriter.AddDir (1359), TTarWriter.AddString (1359), TTarWriter.AddVolumeHeader (1360)

48.6.11 TTarWriter.AddLink

Synopsis: Add hard link to archive.

Declaration:
```
procedure AddLink(Filename: AnsiString; Linkname: AnsiString;
                  DateGmt: TDateTime)
```

Visibility: public

Description: `AddLink` adds a hard link entry to the archive. The entry has name `FileName`, timestamp `DateGmt` and points to `LinkName`.

The link entry is written with the current owner data and permissions.

See also: TTarWriter.AddFile (1358), TTarWriter.AddStream (1359), TTarWriter.AddSymbolicLink (1360), TTarWriter.AddDir (1359), TTarWriter.AddString (1359), TTarWriter.AddVolumeHeader (1360)

48.6.12 TTarWriter.AddVolumeHeader

Synopsis: Add volume header entry.

Declaration:
```
procedure AddVolumeHeader(VolumeId: AnsiString; DateGmt: TDateTime)
```

Visibility: public

Description: `AddVolumeHeader` adds a volume header entry to the archive. The entry is written with name `VolumeID` and timestamp `DateGmt`.

The volume header entry is written with the current owner data and permissions.

See also: TTarWriter.AddFile (1358), TTarWriter.AddStream (1359), TTarWriter.AddSymbolicLink (1360), TTarWriter.AddDir (1359), TTarWriter.AddString (1359), TTarWriter.AddLink (1360)

48.6.13 TTarWriter.Finalize

Synopsis: Finalize the archive.

Declaration:
```
procedure Finalize
```

Visibility: public

Description: `Finalize` writes the end-of-archive marker to the archive. No more entries can be added after `Finalize` was called.

If the `TTarWriter` instance is destroyed, it will automatically call `finalize` if `finalize` was not yet called.

See also: TTarWriter.Destroy (1358)

48.6.14 TTarWriter.Permissions

Synopsis: Archive entry permissions.

Declaration: `Property Permissions : TTarPermissions`

Visibility: public

Access: Read,Write

Description: `Permissions` is used for the permissions field of the archive entries.

See also: TTarDirRec (1353)

48.6.15 TTarWriter.UID

Synopsis: Archive entry user ID.

Declaration: `Property UID : Integer`

Visibility: public

Access: Read,Write

Description: `UID` is used for the `UID` field of the archive entries.

See also: TTarDirRec (1353)

48.6.16 TTarWriter.GID

Synopsis: Archive entry group ID.

Declaration: `Property GID : Integer`

Visibility: public

Access: Read,Write

Description: `GID` is used for the `GID` field of the archive entries.

See also: TTarDirRec (1353)

48.6.17 TTarWriter.UserName

Synopsis: Archive entry user name.

Declaration: `Property UserName : AnsiString`

Visibility: public

Access: Read,Write

Description: `UserName` is used for the `UserName` field of the archive entries.

See also: TTarDirRec (1353)

48.6.18 TTarWriter.GroupName

Synopsis: Archive entry group name.

Declaration: `Property GroupName : AnsiString`

Visibility: public

Access: Read,Write

Description: `GroupName` is used for the `GroupName` field of the archive entries.

See also: TTarDirRec (1353)

48.6.19 TTarWriter.Mode

Synopsis: Archive entry mode.

Declaration: `Property Mode : TTarModes`

Visibility: public

Access: Read,Write

Description: `Mode` is used for the `Mode` field of the archive entries.

See also: TTarDirRec (1353)

48.6.20 TTarWriter.Magic

Synopsis: Archive entry Magic constant.

Declaration: `Property Magic : AnsiString`

Visibility: public

Access: Read,Write

Description: `Magic` is used for the `Magic` field of the archive entries.

See also: TTarDirRec (1353)

Chapter 49

Reference for unit 'MaskUtils'

49.1 Used units

Table 49.1: Used units by unit 'MaskUtils'

Name	Page
System	??
sysutils	??

49.2 Overview

This unit provides routines to work with edit masks. For this, it provied FormatMaskText (1366), FormatMaskInput (1365) and MaskDoFormatText (1366) functions, as well as the underlying TMaskUtils (1366) class.

49.3 Constants, types and variables

49.3.1 Types

```
TEditMask = string
```

`TEditMask` is a string type with different RTTI, this is used to enable registering special property handlers in an IDE.

```
TMaskeditTrimType = (metTrimLeft,metTrimRight)
```

Table 49.2: Enumeration values for type TMaskeditTrimType

Value	Explanation
metTrimLeft	Trimm on the left (start) of the string.
metTrimRight	Trim on the right (end) of the string.

`TMaskeditTrimType` is used when calculating output strings. It is an internal type for the TMAskUtils (1366) class.

metTrimLeft Trimm on the left (start) of the string.

metTrimRight Trim on the right (end) of the string.

```
tMaskedType = (Char_Start,Char_Number,Char_NumberFixed,
  Char_NumberPlusMin,Char_Letter,Char_LetterFixed,
  Char_LetterUpCase,Char_LetterDownCase,
                Char_LetterFixedUpCase
  ,Char_LetterFixedDownCase,
                Char_AlphaNum,Char_AlphaNumFixed
  ,Char_AlphaNumUpCase,
                Char_AlphaNumDownCase,Char_AlphaNumFixedUpCase
  ,
                Char_AlphaNumFixedDownCase,Char_All,Char_AllFixed
  ,
                Char_AllUpCase,Char_AllDownCase,Char_AllFixedUpCase
  ,
                Char_AllFixedDownCase,Char_HourSeparator,
  Char_DateSeparator,Char_Stop)
```

Table 49.3: Enumeration values for type tMaskedType

Value	Explanation
Char_All	Any ASCII or space.
Char_AllDownCase	Any lowercase character.
Char_AllFixed	Any ASCII character.
Char_AllFixedDownCase	Any lowercase character, no space.
Char_AllFixedUpCase	Any uppercase character, no space.
Char_AllUpCase	Any uppercase character.
Char_AlphaNum	One of A-Z, a-z, 0-9 or space.
Char_AlphaNumDownCase	One of a-z, 0-9 or space.
Char_AlphaNumFixed	One of A-Z, a-z, 0-9.
Char_AlphaNumFixedDownCase	One of a-z, 0-9.
Char_AlphaNumFixedUpCase	One of A-Z, 0-9.
Char_AlphaNumUpCase	One of A-Z, 0-9 or space.
Char_DateSeparator	The system locale date separator.
Char_HourSeparator	The system locale hour separator.
Char_Letter	One of A-Z, a-z or a space.
Char_LetterDownCase	One of a-z or space.
Char_LetterFixed	One of A-Z, a-z.
Char_LetterFixedDownCase	One of a-z.
Char_LetterFixedUpCase	One of A-Z.
Char_LetterUpCase	One of A-Z or space.
Char_Number	Numerical character or space.
Char_NumberFixed	Numerical character.
Char_NumberPlusMin	Numerical character, plus or minus or space.
Char_Start	Sentinel value: start of enumerate.
Char_Stop	Sentinel value: end of enumerate.

`tMaskedType` is used internally in TMaskUtils (1366) to describe the mask characters.

Char_start Sentinel value: start of enumerate.

Char_Number Numerical character or space.

Char_NumberFixed Numerical character.

Char_NumberPlusMin Numerical character, plus or minus or space.

Char_Letter One of A-Z, a-z or a space.

Char_LetterFixed One of A-Z, a-z.

Char_LetterUpCase One of A-Z or space.

Char_LetterDownCase One of a-z or space.

Char_LetterFixedUpCase One of A-Z.

Char_LetterFixedUpCase One of A-Z.

Char_LetterFixedDownCase One of a-z.

Char_AlphaNum One of A-Z, a-z, 0-9 or space.

Char_AlphaNumFixed One of A-Z, a-z, 0-9.

Char_AlphaNumUpCase One of A-Z, 0-9 or space.

Char_AlphaNumDownCase One of a-z, 0-9 or space.

Char_AlphaNumFixedUpCase One of A-Z, 0-9.

Char_AlphaNumFixedDownCase One of a-z, 0-9.

Char_All Any ASCII or space.

Char_AllFixed Any ASCII character.

Char_AllUpCase Any uppercase character.

Char_AllDownCase Any lowercase character.

Char_AllFixedUpCase Any uppercase character, no space.

Char_AllFixedDownCase Any lowercase character, no space.

Char_HourSeparator The system locale hour separator.

Char_DateSeparator The system locale date separator.

49.4 Procedures and functions

49.4.1 FormatMaskInput

Synopsis: Return an input mask text based on the edit mask.

Declaration: `function FormatMaskInput(const EditMask: string) : string`

Visibility: default

Description: `FormatMaskInput` returns a text which corresponds to an empty value for `EditMask`. This can e.g. be displayed in an edit box, when the user has not yet typed anything.

See also: FormatMaskText (1366), MaskDoFormatText (1366), TMaskUtils (1366), TMaskUtils.InputMask (1368)

49.4.2 FormatMaskText

Synopsis: Format a text according to a given mask.

Declaration:
```
function FormatMaskText(const EditMask: string; const AValue: string)
                        : string
```

Visibility: default

Description: `FormatMaskText` formats `aValue` according to the mask placeholders found in `EditMask` and returns the resulting string. It replaces space characters in the format mask with the actual space character. This function uses an TMaskUtils (1366) instance to do the actual work.

Errors: None.

See also: FormatMaskInput (1365), MaskDoFormatText (1366), TMaskUtils (1366), TMaskUtils.ApplyMaskToText (1366), TMaskUtils.GetTextWithoutSpaceChar (1366)

49.4.3 MaskDoFormatText

Synopsis: Return an input mask text without literals or space char.

Declaration:
```
function MaskDoFormatText(const EditMask: string; const AValue: string;
                          ASpaceChar: Char) : string
```

Visibility: default

Description: `MaskDoFormatText` formats the `aValue` string using `EditMask`, but enforces the 2d (space char is `aSpaceChar`) and 3d (force use of literals) fields of `EditMask`.

See also: FormatMaskInput (1365), FormatMaskText (1366), TMaskUtils (1366), TMaskUtils.InputMask (1368)

49.5 TMaskUtils

49.5.1 Description

`TEditMask` can be used to work with edit masks. It is used in the FormatMaskInput (1365), FormatMaskText (1366) and MaskDoFormatText (1366) calls to do the actual work.

In general, the work is done by setting the Mask (1367) and Values (1367) properties, and reading the InputMask (1367) property or calling one of ValidateInput (1367) or TryValidateInput (1367).

See also: Mask (1367), Values (1367), InputMask (1367), ValidateInput (1367), TryValidateInput (1367)

49.5.2 Method overview

Page	Method	Description
1367	TryValidateInput	
1367	ValidateInput	Check that the `Value` text is valid for the given mask.

49.5.3 Property overview

Page	Properties	Access	Description
1368	InputMask	r	Text to be used as input mask.
1367	Mask	rw	The edit mask to use.
1367	Value	rw	(input) String value to validate.

49.5.4 TMaskUtils.ValidateInput

Synopsis: Check that the `Value` text is valid for the given mask.

Declaration: `function ValidateInput : string`

Visibility: public

Description: `ValidateInput` checks that the text in Value (1367) satisfies the given mask in Mask (1367) and returns the value with the mask applied to it. If `Value` does not satisfy the mask, an exception is raised. If you don't want an exception, use TryValidateInput (1367) instead.

Errors: If `Value` does not satisfy the mask, an Exception (1363) is raised.

See also: TryValidateInput (1367), Value (1367), Mask (1367), Exception (1363)

49.5.5 TMaskUtils.TryValidateInput

Synopsis:

Declaration: `function TryValidateInput(out ValidatedString: string) : Boolean`

Visibility: public

Description: `TryValidateInput` checks that the text in Value (1367) satisfies the given mask in Mask (1367) and returns `True` if it does, `False` otherwise. It returns `Value` with the mask applied to it in `ValidatedString`.

See also: ValidateInput (1367), Value (1367), Mask (1367)

49.5.6 TMaskUtils.Mask

Synopsis: The edit mask to use.

Declaration: `Property Mask : string`

Visibility: public

Access: Read,Write

Description: `Mask` is the edit mask which must be used when validating Value (1367).

See also: Value (1367), InputMask (1368)

49.5.7 TMaskUtils.Value

Synopsis: (input) String value to validate.

Declaration: `Property Value : string`

Visibility: public

Access: Read,Write

Description: `Value` is the string value which is being validated with Mask (1367).

See also: Mask (1367), InputMask (1368)

49.5.8 TMaskUtils.InputMask

Synopsis: Text to be used as input mask.

Declaration: `Property InputMask : string`

Visibility: public

Access: Read

Description: `InputMask` is a text that can be used as text in an edit value when there is no input: this is the input mask.

See also: Mask (1367), Value (1367)

Chapter 50

Reference for unit 'memds'

50.1 Used units

Table 50.1: Used units by unit 'memds'

Name	Page
Classes	??
DB	438
System	??
sysutils	??
Types	??

50.2 Overview

memds.pp contains classes, types, and routines needed to implement TMemDataset, an in-memory dataset. Ideas implemented in TMemDataset were taken from the THKMemTab component by Harri Kasulke. (Hamburg/Germany)

50.3 Constants, types and variables

50.3.1 Constants

```
MarkerSize = SizeOf(Integer)
```

MarkerSize is a constant that indicates the size for markers used in TMemDataset. Markers are read from and written to the internal TMemoryStream for the in-memory dataset, and separates field definitions from the record data in the stream. A marker is also used to indicate the end of the stream.

MarkerSize is defined as the size for the Integer data type.

```
smData = 2
```

smData is an Integer constant that contains the marker used to signify the start of record data for an in-memory dataset. smData is used in TMemDataset methods which read or write record values using the stream for the in-memory dataset. The value for smData is 2.

```
smEOF = 0
```

smEOF is an Integer constant that contains the marker used as the End-of-File marker for an in-memory dataset. smEOF is used in TMemDataset methods which read or write data using a file or a stream. The value for smEOF is 0 (zero).

```
smFieldDefs = 1
```

smFieldDefs is an Integer constant that contains the marker used to signify the start of field definitions for an in-memory dataset. smFieldDefs is used in TMemDataset methods which read or write field definitions for the in-memory dataset. The value for smFieldDefs is 1.

50.4 MDSError

50.4.1 Description

MDSError is an Exception type raised when an error occurs while reading or writing values for an in-memory dataset. MDSError is raised in the TMemDataset.RaiseError method and uses messages defined in resource strings in the implementation for the unit, including:

- Fieldtype of Field "%s" not supported
- Bookmark %d not found
- Error in data stream at position %d
- Wrong data stream marker at position %d. Got %d, expected %d'
- Filename must not be empty

An MDSError exception will be raised when a field definition uses a data type not supported in TMemDataset. The exception will be raised for the following field types:

- ftADT
- ftCursor
- ftDataSet
- ftDBaseOle
- ftFmtMemo
- ftGraphic
- ftIDispatch
- ftInterface
- ftOraBlob
- ftOraClob
- ftParadoxOle
- ftReference
- ftTimeStamp

- ftTypedBinary
- ftVariant
- ftUnknown

See also: TMemDataset.FieldDefs (1380), TField.DataType (562)

50.5 TMemDataset

50.5.1 Description

TMemDataset is a TDataset descendant which implements an in-memory dataset. TMemDataset is a performant, single user dataset for non-mission critical use cases that do not require transactions. All record and field processing is done in memory; no data is read from or written to disk unless explicitly requested.

TMemDataset implements common facilities defined in the TDataset ancestor class. This includes using the FieldDefs property to define the structure for the dataset. Most (but not all) field types are supported in TMemDataset, including:

- ftString
- ftGuid
- ftFixedChar
- ftBoolean
- ftCurrency
- ftFloat
- ftBCD
- ftLargeInt
- ftSmallInt
- ftWord
- ftInteger
- ftAutoInc (behave like ftInteger i.e. no auto-increment functionality)
- ftDateTime
- ftDate
- ftTime
- ftFmtBCD
- ftWideString
- ftFixedWideChar
- ftBytes
- ftVarBytes

- ftBlob
- ftMemo
- ftWideMemo

TMemDataset implements common data manipulation methods such as: Append, AppendRecord, Insert, InsertRecord, Delete, Clear, and Refresh. TMemDataset implements Bookmarks and common navigation methods like: First, Next, Prior, Last, Locate, BOF, and EOF. Methods are provided that allow loading and saving both structure and data from a file, a stream, or another TDataset descendent.

TMemDataset provides methods to filter records, but they are implemented in a different manner than in TDataset. The Filter property is ignored; use the OnFilterRecord method and the Filtered property for this functionality.

One notable missing feature is Indexes. Index definitions are not implemented in TMemDataset.

TMemDataset uses ideas taken from the THKMemTab component by Harri Kasulke. (Hamburg/Germany)

See also: TDataset (496)

50.5.2 Method overview

Page	Method	Description
1374	BookmarkValid	Determines if the specified Bookmark is valid.
1376	Clear	Clears the content in the in-memory dataset.
1374	CompareBookmarks	Gets the relative order for the specified Bookmarks.
1378	CopyFromDataset	Loads field definitions and optional data from the specified TDataset.
1373	Create	Constructor for the class instance.
1374	CreateBlobStream	Creates a stream used to read or write Blob field data in the in-memory dataset.
1375	CreateTable	Creates the internal storage for records in the in-memory dataset.
1376	DataSize	Size of the internal TMemoryStream used in the in-memory dataset.
1373	Destroy	Destructor for the class instance.
1377	LoadFromFile	Loads the content for the dataset from the specified file name.
1377	LoadFromStream	Loads the content for the dataset from the specified stream.
1375	Locate	Locates a record with the specified values in the in-memory dataset.
1375	Lookup	Searches for a record with the specified values, and returns a list of values.
1376	SaveToFile	Saves field definitions and optional record data to the specified file name.
1377	SaveToStream	Saves field definitions and optional record data to the specified stream.

50.5.3 Property overview

Page	Properties	Access	Description
1380	Active		Indicates if the in-memory dataset is Active.
1382	AfterCancel		
1381	AfterClose		
1382	AfterDelete		
1381	AfterEdit		
1381	AfterInsert		
1380	AfterOpen		
1382	AfterPost		
1383	AfterScroll		
1382	BeforeCancel		
1381	BeforeClose		
1382	BeforeDelete		
1381	BeforeEdit		
1381	BeforeInsert		
1380	BeforeOpen		
1382	BeforePost		
1383	BeforeScroll		
1380	FieldDefs		Field definitions for the in-memory dataset.
1379	FileModified	r	Indicates if the in-memory dataset has been modified.
1379	FileName	rw	File name used to read or write field definitions and optional data.
1379	Filter		Filter for the dataset.
1380	Filtered		Indicates if records in the dataset are filtered using OnFilterRecord.
1383	OnDeleteError		
1383	OnEditError		
1384	OnFilterRecord		Event handler signalled to include or exclude records in the in-memory dataset.
1383	OnNewRecord		
1384	OnPostError		

50.5.4 TMemDataset.Create

Synopsis: Constructor for the class instance.

Declaration: `constructor Create(AOwner: TComponent);  Override`

Visibility: public

Description: Create is the overridden constructor for the class instance. Create calls the inherited constructor.

Create allocates resources required for internal member variables in the class, such as the TMemoryStream that contains the field definitions and record data and the list used to store Blob data. Other internal member variables are set to their default values. Creates sets the default values for the following published properties:

BookmarkSize SizeOf(LongInt)

50.5.5 TMemDataset.Destroy

Synopsis: Destructor for the class instance.

Declaration: `destructor Destroy;  Override`

Visibility: public

Description: Destroy is the overridden destructor for the class instance. Destroy frees resources allocated to internal member variables in the class. The list used for Blob data is cleared and freed. Destroy calls the inherited destructor, and frees the internal memory stream for the class.

50.5.6 TMemDataset.BookmarkValid

Synopsis: Determines if the specified Bookmark is valid.

Declaration:
```
function BookmarkValid(ABookmark: TBookMark) : Boolean;  Override
```

Visibility: public

Description: BookmarkValid is an overridden Boolean function used to determine if the specified Bookmark is valid. BookmarkValid implements the virtual method defined in the ancestor class.

In TMemDataset, a Bookmark is considered to be valid when it contains an Integer value that represents a record in the in-memory dataset. Bookmarks are zero-based and must be less than the record count for dataset.

The return value is False when ABookmark is unassigned (contains `Nil`), or True when the preceding conditions are satisfied.

50.5.7 TMemDataset.CompareBookmarks

Synopsis: Gets the relative order for the specified Bookmarks.

Declaration:
```
function CompareBookmarks(Bookmark1: TBookMark; Bookmark2: TBookMark)
                                : LongInt;  Override
```

Visibility: public

Description: CompareBookmarks is an overridden LongInt function which determines the relative order for the specified Bookmarks.

50.5.8 TMemDataset.CreateBlobStream

Synopsis: Creates a stream used to read or write Blob field data in the in-memory dataset.

Declaration:
```
function CreateBlobStream(Field: TField; Mode: TBlobStreamMode)
                                : TStream;  Override
```

Visibility: public

Description: CreateBlobStream is an overridden TStream function which creates a TMDSBlobStream for the specified field with the read/write permissions in Mode. CreateBlobStream is called when the specified Field needs to read or write its value (for TBlobField or descendent field types).

Mode indicates the permissions need for the blob stream. When Mode contains bmWrite, the value in the State property must indicate that the editing operation is enabled. An exception is raised using DatabaseErrorFmt if State contains a value other than: dsEdit, dsInsert, dsFilter, or dsCalcFields.

Similarly, the Field must allow editing when not in dsSetKey or dsFilter state. An exception is raised using DatabaseErrorFmt if Field has its ReadOnly property set.

The return value contains the TMDSBlobStream instance created using Field and Mode as arguments. Please note that the Blob stream is not saved as part of the data in the in-memory dataset; in its dynamically created and freed as needed.

See also: TFieldDef.DataType (580), TDataset.State (524), TField.ReadOnly (573)

50.5.9 TMemDataset.Locate

Synopsis: Locates a record with the specified values in the in-memory dataset.

Declaration:
```
function Locate(const KeyFields: string; const KeyValues: Variant;
                  Options: TLocateOptions) : Boolean;  Override
```

Visibility: public

Description: Locate is an overridden Boolean function used to locate a record with the specified values in the specified fields. LocateOptions indicates if case-insensitivity or partial keys searches are used in the method. Locate calls the inherited method to ensure that the dataset is bi-directional. Locate calls CheckActive to ensure that the dataset has been opened prior to searching for values in record data. Locate calls MDSLocateRecord to get the return value for the method. When the return value is True, the current record for the dataset is updated and Resync is called to update the active record buffer.

See also: TDataset.Locate (513), TLocateOption (450), TLocateOptions (450), TDataset.Resync (516)

50.5.10 TMemDataset.Lookup

Synopsis: Searches for a record with the specified values, and returns a list of values.

Declaration:
```
function Lookup(const KeyFields: string; const KeyValues: Variant;
                  const ResultFields: string) : Variant;  Override
```

Visibility: public

Description: Lookup is an overridden Variant function used to search for the first record that matches the specified values. KeyFields is a comma-delimited list of field names to examine in the method. KeyValues is a variant array with values for the specified field names. ResultFields is a comma-delimited list of field names to include in the return values for the method.

Lookup calls MDSLocateRecord to search for the specified values in the record data for the in-memory dataset. If a record is located that matches the search criteria, calculated or lookup fields in the dataset are recalculated. The return value is a variant array with values for the fields specified in ResultFields. The return value is set to Null if a record with the specified search values is not found.

Please note that Lookup does not change the active record in the dataset.

For example:

```
var AResultVals: Variant;
  AResultVals := AMemDS.Lookup('lastname, firstname', VarArrayCreate('Franks', 'Pete
```

See also: TMemDataset.Locate (1375)

50.5.11 TMemDataset.CreateTable

Synopsis: Creates the internal storage for records in the in-memory dataset.

Declaration:
```
procedure CreateTable
```

Visibility: public

Description: CreateTable is used to create the internal storage for records in the in-memory dataset. CreateTable calls CheckInactive to ensure that the dataset is not already opened or Active. CreateTable calls Clear to remove any existing record data in the in-memory dataset. Field definitions are retained. CreateTable calls CalcRecordLayout to determine the record size including Bookmark and BookmarkFlag values. Sets the internal TableIsCreated member to True.

See also: TDataset.Active (526), TMemDataset.Clear (1376), TDataset.FieldDefs (521), CreateTable (1375)

50.5.12 TMemDataset.DataSize

Synopsis: Size of the internal TMemoryStream used in the in-memory dataset.

Declaration:
```
function DataSize : Integer
```

Visibility: public

Description: DataSize is an Integer function used to get the size of the internal stream in the in-memory dataset.

50.5.13 TMemDataset.Clear

Synopsis: Clears the content in the in-memory dataset.

Declaration:
```
procedure Clear(ClearDefs: Boolean)
procedure Clear
```

Visibility: public

Description: Clear is an overloaded procedure used to clear record data, Blob streams, and optionally Field definitions in the in-memory dataset. Clear removes any Blob streams allocated for memo fields in the dataset. Clear removes any memory allocated to the internal TMemoryStream used for record data in the dataset. If the dataset is Active, the Resync method is called to refresh values in the active record buffer.

ClearDefs indicates if the FieldDefs for the dataset are also cleared. When ClearDefs is True, the Close method is called to deactivate the dataset. All field definitions in FieldDefs are removed. The internal member TableIsCreated is set to False.

See also: TDataset.Active (526), TDataset.Close (505), TDataset.FieldDefs (521)

50.5.14 TMemDataset.SaveToFile

Synopsis: Saves field definitions and optional record data to the specified file name.

Declaration:
```
procedure SaveToFile(const AFileName: string)
procedure SaveToFile(const AFileName: string; SaveData: Boolean)
```

Visibility: public

Description: SaveToFile is an overloaded procedure used to store field definitions and optional record data in the dataset to the specified file name. AFileName is the file name on the local file system used to store values from the dataset. SaveData indicates if record data is included in the values stored to the file. When SaveData contains False, only the field definitions for the dataset are stored in the file.

AFileName must contain a file name for the local file system. SaveToFile calls RaiseError to raise an exception if the value in AFileName is an empty string (''). SaveToFile creates a TFileStream for the specified file name, and calls SaveToStream to store the content from the dataset.

SaveToFile reimplements the method defined in the ancestor class.

50.5.15 TMemDataset.SaveToStream

Synopsis: Saves field definitions and optional record data to the specified stream.

Declaration:
```
procedure SaveToStream(F: TStream)
procedure SaveToStream(F: TStream; SaveData: Boolean)
```

Visibility: public

Description: SaveToStream is used to save field definitions and optional record data for the in-memory dataset to the specified stream. SaveToStream calls SaveFieldDefsToStream to save the field definitions in FieldDefs to the stream specified in F.

SaveData indicates if record data is included in the values written to the stream. When SaveData contains True, the SaveDataToStream method is called to save record data to the stream. No record data is written when SaveData is False. SaveToStream calls WriteMarker to write the smEOF marker value that signifies the end of record data in the stream.

Use LoadFromStream to load field definitions and record data for the in-memory dataset.

See also: TMemDataset.LoadFromStream (1377)

50.5.16 TMemDataset.LoadFromStream

Synopsis: Loads the content for the dataset from the specified stream.

Declaration:
```
procedure LoadFromStream(F: TStream)
```

Visibility: public

Description: LoadFromStream is used to load the content for the dataset from the specified stream. F is a TStream descendent that is used to load the field definitions and record data for the in-memory dataset. LoadFromStream calls Close to ensure that the dataset saves its existing content (when FileName has been assigned) and clears any default Fields created when the dataset was opened.

LoadFromStream calls ReadFieldDefsFromStream to load field definitions from the stream in F. CreateTable is called to initialize storage for record data in the dataset. LoadDataFromStream is called to load any record data present in the stream. CheckMarker is called to ensure that the stream is positioned on the smEOF marker that signals the end of record data in the stream. An exception is raised if the stream was truncated or does not contain the value smEOF at the current position in the stream. LoadFromStream sets the value in the FileModified property to False.

Use SaveToStream to write the field definitions and record data in the dataset to a stream.

See also: TMemDataset.SaveToStream (1377)

50.5.17 TMemDataset.LoadFromFile

Synopsis: Loads the content for the dataset from the specified file name.

Declaration:
```
procedure LoadFromFile(const AFileName: string)
```

Visibility: public

Description: LoadFromFile is used to load the content for the dataset from the specified file name. LoadFromFile creates a TFileStream for the file name specified in AFileName. The file stream is passed to LoadFromStream to load the contents of the file into the in-memory dataset. The file stream is freed prior to exiting from the method.

Use SaveToFile to save the contents of an in-memory dataset to a file on the local file system.

See also: TMemDataset.LoadFromStream (1377), TMemDataset.SaveToFile (1376)

50.5.18 TMemDataset.CopyFromDataset

Synopsis: Loads field definitions and optional data from the specified TDataset.

Declaration:
```
procedure CopyFromDataset(DataSet: TDataSet)
procedure CopyFromDataset(DataSet: TDataSet; CopyData: Boolean)
```

Visibility: public

Description: CopyFromDataset is used to load field definitions and optional record data from the specified TDataset descendent. Dataset contains the TDataset used as the source for the structure and optional record data loaded in the method. CopyData indicates if record data is loaded in the method. When CopyData contains False, only the structure from Dataset is loaded in method.

CopyFromDataset removes any existing field definitions in FieldDefs, and any record data stored in the in-memory dataset. CopyFromDataset uses the Fields in the DataSet argument to determine the new structure for the in-memory dataset. This is done because the visible Fields in the dataset may differ from the actual field definitions. CopyFromDataset creates and adds a TFieldDef instance to FieldsDefs for each of the Fields in DataSet.

CopyFromDataset calls CreateTable to allocated record storage for the new field definitions in FieldDefs.

When CopyData contains True, record data from the DataSet argument is added to the in-memory dataset. When CopyData contains False, record data in the DataSet argument is ignored.

The Open method is called to activate both datasets. DisableControls is called for both TDatasets to prevent updates during record navigation. All records in DataSet are loaded into the in-memory dataset by calling Append and setting the value for each of the field definitions in the target. Field definitions with the following data types are loaded using the native type for the field:

- ftFixedChar
- ftString
- ftBoolean
- ftFloat
- ftLargeInt
- ftSmallInt
- ftInteger
- ftDate
- ftTime
- ftDateTime

All other field values are loaded using their AsString representation.

CopyFromDataset calls Post after adding each record in the dataset. If an exception occurs, the Cancel method is called and the exception is re-raised.

CopyFromDataset calls the EnableControls method in both datasets when record data has been loaded in the method. Please note that the record position in the DataSet argument is restored after loading record data.

See also: TDataset.Fields (524), TDataset.FieldDefs (521), TMemDataset.CreateTable (1375), TDataset.Append (503), TDataset.Post (515)

50.5.19 TMemDataset.FileModified

Synopsis: Indicates if the in-memory dataset has been modified.

Declaration: `Property FileModified : Boolean`

Visibility: public

Access: Read

Description: FileModified is a read-only Boolean property which indicates if the in-memory dataset has been modified. The value in FileModified is updated in methods that write record buffers to the internal memory stream for the dataset, such as:

- MDSWriteRecord
- MDSAppendRecord
- InternalDelete

The value in FileModified is also updated in methods called when opening or closing the in-memory dataset, such as:

- LoadFromStream
- SaveDataToStream
- InternalClose

50.5.20 TMemDataset.Filter

Synopsis: Filter for the dataset.

Declaration: `Property Filter : ; unimplemented;`

Visibility: public

Access:

Description: Filter is not implemented in TMemDataset. Values assigned to the Filter property are silently discarded. Use OnFilterRecord and Filtered instead.

See also: TMemDataset.Filtered (1380), TMemDataset.OnFilterRecord (1384)

50.5.21 TMemDataset.FileName

Synopsis: File name used to read or write field definitions and optional data.

Declaration: `Property FileName : string`

Visibility: published

Access: Read,Write

Description: FileName is a String property that specifies the file used to load field definitions and optional data when the dataset is opened. When FileName is assigned, and the dataset has been modified, it indicates the file name used to store field definitions and data in the local file system.

50.5.22 TMemDataset.Filtered

Synopsis: Indicates if records in the dataset are filtered using OnFilterRecord.

Declaration: `Property Filtered :`

Visibility: published

Access:

Description: Filtered is a published Boolean property that indicates if records in the dataset are filtered using the OnFilterRecord event handler. Filtered is used methods that retrieve record buffers or perform record searches, and determines if records are visible in the dataset.

When Filtered contains True, the MDSFilterRecord method is called to perform filtering for records in the dataset. Unlike the ancestor class, the Filter property is not used in TMemDataset. Values assigned to the Filter property are silently discarded. Use the OnFilterRecord event handler to implement comparisons need to determine record visibility.

See also: TMemDataset.OnFilterRecord (1384), TDataset.Filter (525)

50.5.23 TMemDataset.Active

Synopsis: Indicates if the in-memory dataset is Active.

Declaration: `Property Active :`

Visibility: published

Access:

50.5.24 TMemDataset.FieldDefs

Synopsis: Field definitions for the in-memory dataset.

Declaration: `Property FieldDefs :`

Visibility: published

Access:

50.5.25 TMemDataset.BeforeOpen

Synopsis:

Declaration: `Property BeforeOpen :`

Visibility: published

Access:

50.5.26 TMemDataset.AfterOpen

Synopsis:

Declaration: `Property AfterOpen :`

Visibility: published

Access:

50.5.27 TMemDataset.BeforeClose

Synopsis:

Declaration: `Property BeforeClose :`

Visibility: published

Access:

50.5.28 TMemDataset.AfterClose

Synopsis:

Declaration: `Property AfterClose :`

Visibility: published

Access:

50.5.29 TMemDataset.BeforeInsert

Synopsis:

Declaration: `Property BeforeInsert :`

Visibility: published

Access:

50.5.30 TMemDataset.AfterInsert

Synopsis:

Declaration: `Property AfterInsert :`

Visibility: published

Access:

50.5.31 TMemDataset.BeforeEdit

Synopsis:

Declaration: `Property BeforeEdit :`

Visibility: published

Access:

50.5.32 TMemDataset.AfterEdit

Synopsis:

Declaration: `Property AfterEdit :`

Visibility: published

Access:

50.5.33 TMemDataset.BeforePost

Synopsis:

Declaration: `Property BeforePost :`

Visibility: published

Access:

50.5.34 TMemDataset.AfterPost

Synopsis:

Declaration: `Property AfterPost :`

Visibility: published

Access:

50.5.35 TMemDataset.BeforeCancel

Synopsis:

Declaration: `Property BeforeCancel :`

Visibility: published

Access:

50.5.36 TMemDataset.AfterCancel

Synopsis:

Declaration: `Property AfterCancel :`

Visibility: published

Access:

50.5.37 TMemDataset.BeforeDelete

Synopsis:

Declaration: `Property BeforeDelete :`

Visibility: published

Access:

50.5.38 TMemDataset.AfterDelete

Synopsis:

Declaration: `Property AfterDelete :`

Visibility: published

Access:

50.5.39 TMemDataset.BeforeScroll

Synopsis:

Declaration: `Property BeforeScroll :`

Visibility: published

Access:

50.5.40 TMemDataset.AfterScroll

Synopsis:

Declaration: `Property AfterScroll :`

Visibility: published

Access:

50.5.41 TMemDataset.OnDeleteError

Synopsis:

Declaration: `Property OnDeleteError :`

Visibility: published

Access:

50.5.42 TMemDataset.OnEditError

Synopsis:

Declaration: `Property OnEditError :`

Visibility: published

Access:

Description:

See also: (**??**)

50.5.43 TMemDataset.OnNewRecord

Synopsis:

Declaration: `Property OnNewRecord :`

Visibility: published

Access:

50.5.44 TMemDataset.OnPostError

Synopsis:

Declaration: `Property OnPostError :`

Visibility: published

Access:

50.5.45 TMemDataset.OnFilterRecord

Synopsis: Event handler signalled to include or exclude records in the in-memory dataset.

Declaration: `Property OnFilterRecord :`

Visibility: published

Access:

Description: OnFilterRecord is a published TFilterRecordEvent property which provides the event handler signalled to include or exclude records in the in-memory dataset. OnFilterRecord provides a way for the application to decide whether a record is visible in the dataset on a record-by-record basis. Applications must assign a procedure to the event handler that performs any comparison needed to determine record visibility. The procedure must set the value in Accept to True to make the record visible in the in-memory dataset.

Set the Filtered property to True to enable the OnFilterRecord event handler during record navigation.

OnFilterRecord is used as an alternative filtering mechanism; TMemDataset does not implement the Filter property. Values assigned to the Filter property are silently discarded.

See also: TDataSet.OnFilterRecord (534), TFilterRecordEvent (449), TDataSet.Filtered (525), TDataSet.Resync (516)

Chapter 51

Reference for unit 'MSSQLConn'

51.1 Used units

Table 51.1: Used units by unit 'MSSQLConn'

Name	Page
BufDataset	213
Classes	??
DB	438
dblib	??
SQLDB	1487
System	??
sysutils	??

51.2 Overview

Connector to Microsoft SQL Server databases. Needs FreeTDS dblib library.

51.3 Constants, types and variables

51.3.1 Variables

```
DBLibLibraryName : string = DBLIBDLL
```

`DBLibLibraryName` is the name of the library to load when dynamically loading support for MS SQL or Sybase. It must be set before the first connection is made.

51.4 EMSSQLDatabaseError

51.4.1 Description

Sybase/MS SQL Server specific error.

51.4.2 Property overview

Page	Properties	Access	Description
1386	DBErrorCode	r	Sybase/MS SQL Server error code.

51.4.3 EMSSQLDatabaseError.DBErrorCode

Synopsis: Sybase/MS SQL Server error code.

Declaration: `Property DBErrorCode : Integer; deprecated;`

Visibility: public

Access: Read

Description: Error code as generated by the database server.

51.5 TMSSQLConnection

51.5.1 Description

Connector to Microsoft SQL Server databases.

Requirements:

MS SQL Server Client Library is required (ntwdblib.dll)

- or -

FreeTDS (dblib.dll)

Older FreeTDS libraries may require freetds.conf: (http://www.freetds.org/userguide/freetdsconf.htm)

[global]

tds version = 7.1

client charset = UTF-8

port = 1433 or instance = ... (optional)

dump file = freetds.log (optional)

text size = 2147483647 (optional)

Known problems:

- CHAR/VARCHAR data truncated to column length when encoding to UTF-8 (use NCHAR/NVARCHAR instead or CAST char/varchar to nchar/nvarchar)

- Multiple result sets (MARS) are not supported (for example when SP returns more than 1 result set only 1st is processed)

- DB-Library error 10038 "Results Pending": set TSQLQuery.PacketRecords=-1 to fetch all pending rows

- BLOB data (IMAGE/TEXT columns) larger than 16MB are truncated to 16MB: (set TMSSQL-Connection.Params: 'TEXTSIZE=2147483647' or execute 'SET TEXTSIZE 2147483647')

51.5.2 Method overview

Page	Method	Description
1387	Create	Create a new instance of `TMSSQLConnection`.
1387	CreateDB	Create a new MS SQL database.
1388	DropDB	Drop a MS SQL database.
1387	GetConnectionInfo	Return some information about the connection.

51.5.3 Property overview

Page	Properties	Access	Description
1389	CharSet		
1389	Connected		Is the connection active.
1390	DatabaseName		
1389	HostName		Host and optionally port or instance.
1390	KeepConnection		Keep connection alive.
1390	LoginPrompt		Show login prompt.
1391	OnLogin		Called when logging in.
1390	Params		
1388	Password		
1389	Role		Role for user.
1388	Transaction		Default transaction.
1388	UserName		

51.5.4 TMSSQLConnection.Create

Synopsis: Create a new instance of `TMSSQLConnection`.

Declaration: `constructor Create(AOwner: TComponent); Override`

Visibility: public

Description: `Create` is the default constructor for the `TMSSQLConnection` class. It calls the inherited constructor and sets some defaults.

51.5.5 TMSSQLConnection.GetConnectionInfo

Synopsis: Return some information about the connection.

Declaration: `function GetConnectionInfo(InfoType: TConnInfoType) : string; Override`

Visibility: public

Description: `GetConnectionInfo` overrides TSQLConnection.GetConnectionInfo (1518) to return the relevant information for the Interbase/Firebird connection.

See also: TSQLConnection.GetConnectionInfo (1518), TConnInfoType (1497)

51.5.6 TMSSQLConnection.CreateDB

Synopsis: Create a new MS SQL database.

Declaration: `procedure CreateDB; Override`

Visibility: public

Description: `CreateDB` creates a database on the server with given `DatabaseName`.

See also: TMSSQLConnection.DropDB (1388)

51.5.7 TMSSQLConnection.DropDB

Synopsis: Drop a MS SQL database.

Declaration: `procedure DropDB;  Override`

Visibility: public

Description: `DropDB` drops a database on the server with given `DatabaseName`

See also: TMSSQLConnection.CreateDB (1387)

51.5.8 TMSSQLConnection.Password

Declaration: `Property Password :`

Visibility: published

Access:

Description: `TMSSQLConnection` specific: if you don't enter a UserName and Password, the connector will try to use Trusted Authentication/SSPI (on Windows only).

51.5.9 TMSSQLConnection.Transaction

Synopsis: Default transaction.

Declaration: `Property Transaction :`

Visibility: published

Access:

Description: `Transaction` is redeclared from TSQLConnection.Transaction (1385)

See also: TSQLConnection.Transaction (1385)

51.5.10 TMSSQLConnection.UserName

Declaration: `Property UserName :`

Visibility: published

Access:

Description: `TMSSQLConnection` specific: if you don't enter a UserName and Password, the connector will try to use Trusted Authentication/SSPI (on Windows only).

51.5.11 TMSSQLConnection.CharSet

Declaration: `Property CharSet :`

Visibility: published

Access:

Description: Character Set - if you use Microsoft DB-Lib and set to 'UTF-8' then char/varchar fields will be UTF8Encoded/Decoded.

If you use FreeTDS DB-Lib, then you must compile with iconv support (requires libiconv2.dll) or cast char/varchar to nchar/nvarchar in SELECTs.

51.5.12 TMSSQLConnection.HostName

Synopsis: Host and optionally port or instance.

Declaration: `Property HostName :`

Visibility: published

Access:

Description: `TMSSQLConnection` specific: you can specify an instance or a port after the host name itself.

Instance should be specified with a backslash e.g.: 127.0.0.0.1\SQLEXPRESS. Port should be specified with a colon, e.g. BIGBADSERVER:1433

See http://www.freetds.org/userguide/PortOverride.html

51.5.13 TMSSQLConnection.Connected

Synopsis: Is the connection active.

Declaration: `Property Connected :`

Visibility: published

Access:

Description: `Connected` can be set to `True` to activate the connection, or to `False` to close the connection.

51.5.14 TMSSQLConnection.Role

Synopsis: Role for user.

Declaration: `Property Role :`

Visibility: published

Access:

Description: `Role` is redeclared from TSQLConnection.Role (1385)

51.5.15 TMSSQLConnection.DatabaseName

Declaration: `Property DatabaseName :`

Visibility: published

Access:

Description: `TMSSQLConnection` specific: the `master` database should always exist on a server.

51.5.16 TMSSQLConnection.KeepConnection

Synopsis: Keep connection alive.

Declaration: `Property KeepConnection :`

Visibility: published

Access:

Description: `KeepConnection` is redeclared from TSQLConnection.KeepConnection (1385)

See also: TSQLConnection.KeepConnection (1385)

51.5.17 TMSSQLConnection.LoginPrompt

Synopsis: Show login prompt.

Declaration: `Property LoginPrompt :`

Visibility: published

Access:

Description: `LoginPrompt` is redeclared from TSQLConnection.LoginPrompt (1385)

See also: TSQLConnection.LoginPrompt (1385)

51.5.18 TMSSQLConnection.Params

Declaration: `Property Params :`

Visibility: published

Access:

Description: `TMSSQLConnection` specific:

set "AutoCommit=true" if you don't want to explicitly commit/rollback transactions

set "TextSize=16777216 - to set maximum size of blob/text/image data returned. Otherwise, these large fields may be cut off when retrieving/setting data.

51.5.19 TMSSQLConnection.OnLogin

Synopsis: Called when logging in.

Declaration: `Property OnLogin :`

Visibility: published

Access:

Description: `OnLogin` is redeclared from TSQLConnection.OnLogin (1385)

See also: TSQLConnection.OnLogin (1385)

51.6 TMSSQLConnectionDef

51.6.1 Description

Describes the MS SQL connection properties for TSQLConnector (1385)

See also: TMSSQLConnection (1386), TSQLConnector (1385)

51.6.2 Method overview

Page	Method	Description
1391	ConnectionClass	Connection class to use.
1392	DefaultLibraryName	Default name of the MSSQL client library.
1392	Description	Short description of connection.
1392	LoadedLibraryName	Actally loaded library name.
1392	LoadFunction	Return Function to call when loading MS-SQL support.
1391	TypeName	Connection type name.
1392	UnLoadFunction	Return Function to call when unloading MS-SQL support.

51.6.3 TMSSQLConnectionDef.TypeName

Synopsis: Connection type name.

Declaration: `class function TypeName : string;  Override`

Visibility: default

Description: `TypeName` returns the unique name of the MS-SQL connection.

51.6.4 TMSSQLConnectionDef.ConnectionClass

Synopsis: Connection class to use.

Declaration: `class function ConnectionClass : TSQLConnectionClass;  Override`

Visibility: default

Description: `ConnectionClass` returns TMSSQLConnection (1386)

See also: TMSSQLConnection (1386)

51.6.5 TMSSQLConnectionDef.Description

Synopsis: Short description of connection.

Declaration: `class function Description : string;  Override`

Visibility: default

Description: `Description` describes the MS SQL connector type.

51.6.6 TMSSQLConnectionDef.DefaultLibraryName

Synopsis: Default name of the MSSQL client library.

Declaration: `class function DefaultLibraryName : string;  Override`

Visibility: default

Description: `DefaultLibraryName` returns the library name to use when loading the MSSQL client library.

51.6.7 TMSSQLConnectionDef.LoadFunction

Synopsis: Return Function to call when loading MS-SQL support.

Declaration: `class function LoadFunction : TLibraryLoadFunction;  Override`

Visibility: default

Description: `LoadFunction` is used by the connector logic to get the function to dynamically load MS-SQL support.

51.6.8 TMSSQLConnectionDef.UnLoadFunction

Synopsis: Return Function to call when unloading MS-SQL support.

Declaration: `class function UnLoadFunction : TLibraryUnLoadFunction;  Override`

Visibility: default

Description: `UnLoadFunction` is used by the connector logic to get the function to unload MS-SQL support.

51.6.9 TMSSQLConnectionDef.LoadedLibraryName

Synopsis: Actally loaded library name.

Declaration: `class function LoadedLibraryName : string;  Override`

Visibility: default

Description: `LoadedLibraryName` returns the actually loaded library name.

See also: DefaultLibraryName (1392)

51.7 TSybaseConnection

51.7.1 Description

Connector to Sybase Adaptive Server Enterprise (ASE) database servers.

Requirements:

FreeTDS (dblib.dll)

Older FreeTDS libraries may require freetds.conf: (http://www.freetds.org/userguide/freetdsconf.htm)

[global]

tds version = 7.1

client charset = UTF-8

port = 5000 (optional)

dump file = freetds.log (optional)

text size = 2147483647 (optional)

51.7.2 Method overview

Page	Method	Description
1393	Create	Create a Sybase database connection.

51.7.3 TSybaseConnection.Create

Synopsis: Create a Sybase database connection.

Declaration: `constructor Create(AOwner: TComponent);  Override`

Visibility: public

Description: `Create` is the default constructor for the `TSybaseConnection` class. It calls the inherited constructor and sets some defaults.

51.8 TSybaseConnectionDef

51.8.1 Description

Describes the MS SQL connection properties for TSQLConnector (1385)

See also: TSybaseConnection (1393), TSQLConnector (1385)

51.8.2 Method overview

Page	Method	Description
1394	ConnectionClass	Connection class to use.
1394	Description	Short description of connection.
1393	TypeName	Connection type name.

51.8.3 TSybaseConnectionDef.TypeName

Synopsis: Connection type name.

Declaration: `class function TypeName : string; Override`

Visibility: default

Description: `TypeName` returns the unique name of the Sybase connection.

51.8.4 TSybaseConnectionDef.ConnectionClass

Synopsis: Connection class to use.

Declaration: `class function ConnectionClass : TSQLConnectionClass; Override`

Visibility: default

Description: `ConnectionClass` returns TSybaseConnection (1393)

See also: TSybaseConnection (1393)

51.8.5 TSybaseConnectionDef.Description

Synopsis: Short description of connection.

Declaration: `class function Description : string; Override`

Visibility: default

Description: `Description` describes the Sybase connector type.

Chapter 52

Reference for unit 'nullstream'

52.1 Used units

Table 52.1: Used units by unit 'nullstream'

Name	Page
Classes	**??**
System	**??**

52.2 Overview

The `nullstream` unit implements TNullStream (1395), a stream which acts more or less as the `/dev/null` device on unix: all read and write operations will succeed, but the data is discarded on write, or null bytes are read.

52.3 ENullStreamError

52.3.1 Description

`ENullStreamError` is the exception raised when TNullStream.Seek (1396) results in an invalid position.

See also: TNullStream.Seek (1396)

52.4 TNullStream

52.4.1 Description

`TNullStream` discards any data written to it (but keeps a virtual size) and returns 0 bytes when read from. It emulates a #rtl.classes.TMemoryStream (**??**): When writing to the stream, the size is increased as needed. When reading, the maximum number of returned bytes is limited to the size of the stream.

See also: TNullStream.Read (1396), TNullStream.Write (1396), #rtl.classes.TStream.Size (**??**)

52.4.2 Method overview

Page	Method	Description
1397	Create	Create a new instance.
1396	Read	Read null bytes from the stream.
1396	Seek	Set current position in the stream.
1396	Write	Write to stream.

52.4.3 TNullStream.Read

Synopsis: Read null bytes from the stream.

Declaration:
```
function Read(var Buffer; Count: LongInt) : LongInt;  Override
```

Visibility: public

Description: `Read` reads `Count` null bytes from the stream. `Count` can be at most `Size`. The `Buffer` will be filled with null bytes, effectively zeroing out the memory. The size can be increased using `Write` or by explicitly setting `Size`.

See also: TNullStream.Write (1396), #rtl.classes.TStream.Size (**??**)

52.4.4 TNullStream.Write

Synopsis: Write to stream.

Declaration:
```
function Write(const Buffer; Count: LongInt) : LongInt;  Override
```

Visibility: public

Description: `Write` simulates a write operation: no data is actually written from `Buffer`, but the size of the stream is enlarged if the amount of bytes `Count` and current position in the stream make this necessary.

See also: TNullStream.Read (1396), #rtl.classes.TStream.Size (**??**)

52.4.5 TNullStream.Seek

Synopsis: Set current position in the stream.

Declaration:
```
function Seek(const Offset: Int64; Origin: TSeekOrigin) : Int64
                ;  Override
```

Visibility: public

Description: `Seek` sets the current position in the stream. It simulates this operation by keeping a "virtual" position. See #rtl.classes.TStream.Seek (**??**) for more info about the arguments.

Errors: If the requested operation would cause the position to fall outside of the allowed range (0 to `Size`) then a ENullStreamError (1395) exception is raised.

See also: TNullStream.Read (1396), TNullStream.Write (1396), #rtl.classes.TStream.Seek (**??**)

52.4.6 TNullStream.Create

Synopsis: Create a new instance.

Declaration: `constructor Create`

Visibility: public

Description: `Create` initializes the size and position of the stream to zero.

See also: #rtl.classes.TStream.Position (**??**), #rtl.classes.TStream.Size (**??**)

Chapter 53

Reference for unit 'pascodegen'

53.1 Used units

Table 53.1: Used units by unit 'pascodegen'

Name	Page
Classes	??
System	??
sysutils	??

53.2 Overview

pascodegen is a unit with a class that can be used to generate pascal code. it offers the TPascalCodeGenerator (1399) class which has some utility methods for indenting code, offers logging and automatic handling of code sections or blocks (variable, constant etc.) The resulting code is available in a stringlist.

53.3 Constants, types and variables

53.3.1 Types

```
TCodeGeneratorLogEvent = procedure(Sender: TObject;
  LogType: TCodegenLogType;
  const Msg: string) of object
```

`TCodeGeneratorLogEvent` is an event handler type used for logging messages from the code generator. It receives the sender object, log type and message text.

```
TCodegenLogType = (cltInfo)
```

Table 53.2: Enumeration values for type TCodegenLogType

Value	Explanation
cltInfo	Informational log message

`TCodegenLogType` defines the types of log messages that can be generated by the code generator.

`cltInfo` Informational message

`cltWarning` Warning message

```
TCodegenLogTypes = Set of TCodegenLogType
```

`TCodegenLogTypes` is a set type based on TCodegenLogType (1398) enumeration. It allows specifying which types of log messages should be processed.

```
TCodeSection = (csUnknown,csConst,csType,csVar,csResourcestring,
  csDeclaration)
```

Table 53.3: Enumeration values for type TCodeSection

Value	Explanation
csConst	Constants section
csDeclaration	General declarations section
csResourcestring	Resource strings section
csType	Types section
csUnknown	Unknown section type
csVar	Variables section

`TCodeSection` defines the different types of sections that can appear in Pascal code.

`csUnknown` Unknown or unspecified section type

`csConst` Const section for constants

`csType` Type section for type definitions

`csVar` Var section for variables

`csResourcestring` Resourcestring section for localizable strings

`csDeclaration` General declaration section

53.4 TPascalCodeGenerator

53.4.1 Description

`TPascalCodeGenerator` is a class that provides functionality for generating Pascal code. It offers utility methods for code indentation, section management, logging and automatic handling of code blocks. The resulting generated code is available in a string list accessible through the TPascalCodeGenerator.Source (1405) property. The class supports dotted unit names and provides configurable keyword escaping.

See also: TCodeSection (1399), TCodeGeneratorLogEvent (1398)

53.4.2 Method overview

Page	Method	Description
1404	AddLn	Add line to source code with current indentation
1404	ClassHeader	Add class header comment (deprecated)
1404	Comment	Add comment to source code
1400	Create	Create new Pascal code generator instance
1402	CreateHeader	Create unit header section
1402	CreateUnitClause	Create unit clause
1401	Destroy	Destroy Pascal code generator instance
1401	EnsureSection	Ensure code is in specified section
1403	EscapeKeyWord	Escape Pascal keyword if needed
1402	Indent	Increase indentation level
1403	IsKeyWord	Check if string is Pascal keyword
1403	MakePascalString	Convert string to Pascal string literal
1401	PopSection	Pop section from section stack
1403	PrettyPrint	Format string with proper capitalization
1401	PushSection	Push section onto section stack
1405	SaveToFile	Save generated source code to file
1405	SaveToStream	Save generated source code to stream
1404	SimpleMethodBody	Generate simple method body
1402	Undent	Decrease indentation level

53.4.3 Property overview

Page	Properties	Access	Description
1407	AddTimeStamp	rw	Whether to include timestamp in header
1405	CurrentSection	rw	Current code section type
1406	ExtraUnits	rw	Additional units for uses clause
1407	KeywordPrefix	rw	Prefix for escaped keywords
1407	KeywordSuffix	rw	Suffix for escaped keywords
1406	LicenseText	rw	License text for header comment
1407	OnLog	rw	Event handler for log messages
1406	OutputUnitName	rw	Name of the output unit
1405	Source	r	Generated source code string list
1406	Switches	rw	Compiler switches for the unit

53.4.4 TPascalCodeGenerator.Create

Synopsis: Create new Pascal code generator instance

Declaration: `constructor Create(AOwner: TComponent);  Override`

Visibility: public

Description: `Create` initializes a new `TPascalCodeGenerator` instance. It creates internal string lists for source code, license text and compiler switches. The default compiler switches MODE ObjFPC and H+ are automatically added.

See also: TPascalCodeGenerator.Destroy (1401)

53.4.5 TPascalCodeGenerator.Destroy

Synopsis: Destroy Pascal code generator instance

Declaration: `destructor Destroy;  Override`

Visibility: public

Description: `Destroy` frees all internal string lists and cleans up the `TPascalCodeGenerator` instance.

See also: TPascalCodeGenerator.Create (1400)

53.4.6 TPascalCodeGenerator.EnsureSection

Synopsis: Ensure code is in specified section

Declaration: `procedure EnsureSection(aSection: TCodeSection)`

Visibility: public

Description: `EnsureSection` ensures that the current code section is set to the specified section type. If the current section is different from the requested section, it outputs the appropriate section keyword (Const, Type, Var, Resourcestring) and updates the current section.

See also: TCodeSection (1399), TPascalCodeGenerator.CurrentSection (1405)

53.4.7 TPascalCodeGenerator.PushSection

Synopsis: Push section onto section stack

Declaration: `procedure PushSection(ASection: TCodeSection)`

Visibility: public

Description: `PushSection` pushes a section onto the internal section stack. This allows nesting of sections and later restoration using TPascalCodeGenerator.PopSection (1401).

See also: TPascalCodeGenerator.PopSection (1401), TCodeSection (1399)

53.4.8 TPascalCodeGenerator.PopSection

Synopsis: Pop section from section stack

Declaration: `function PopSection : TCodeSection`

Visibility: public

Description: `PopSection` pops the topmost section from the internal section stack and returns it. If the stack is empty, it returns `csUnknown`.

See also: TPascalCodeGenerator.PushSection (1401), TCodeSection (1399)

53.4.9 TPascalCodeGenerator.CreateHeader

Synopsis: Create unit header section

Declaration: `procedure CreateHeader; Virtual`

Visibility: public

Description: `CreateHeader` generates the interface header section of a Pascal unit. This includes the license text comment, optional timestamp, compiler switches, interface keyword and uses clause with support for dotted unit names.

See also: TPascalCodeGenerator.CreateUnitClause (1402), TPascalCodeGenerator.LicenseText (1406)

53.4.10 TPascalCodeGenerator.CreateUnitClause

Synopsis: Create unit clause

Declaration: `procedure CreateUnitClause; Virtual`

Visibility: public

Description: `CreateUnitClause` generates the unit clause at the beginning of a Pascal unit. It handles dotted unit support by conditionally generating the appropriate unit name.

See also: TPascalCodeGenerator.CreateHeader (1402), TPascalCodeGenerator.OutputUnitName (1406)

53.4.11 TPascalCodeGenerator.Indent

Synopsis: Increase indentation level

Declaration: `procedure Indent`

Visibility: public

Description: `Indent` increases the current indentation level by adding two spaces to the internal indentation string. Subsequent calls to TPascalCodeGenerator.AddLn (1404) will use this increased indentation.

See also: TPascalCodeGenerator.Undent (1402), TPascalCodeGenerator.AddLn (1404)

53.4.12 TPascalCodeGenerator.Undent

Synopsis: Decrease indentation level

Declaration: `procedure Undent`

Visibility: public

Description: `Undent` decreases the current indentation level by removing two spaces from the internal indentation string. If the indentation string is empty, no action is taken.

See also: TPascalCodeGenerator.Indent (1402), TPascalCodeGenerator.AddLn (1404)

53.4.13 TPascalCodeGenerator.IsKeyWord

Synopsis: Check if string is Pascal keyword

Declaration: `function IsKeyWord(const S: string) : Boolean`

Visibility: public

Description: `IsKeyWord` checks if the given string is a Pascal language keyword. It performs a case-insensitive comparison against a complete list of Pascal keywords including reserved words, operators and built-in identifiers.

See also: TPascalCodeGenerator.EscapeKeyWord (1403)

53.4.14 TPascalCodeGenerator.EscapeKeyWord

Synopsis: Escape Pascal keyword if needed

Declaration:
```
function EscapeKeyWord(const S: string; ForceAmpersand: Boolean)
                       : string
```

Visibility: public

Description: `EscapeKeyWord` escapes a string if it is a Pascal keyword by adding prefix and suffix characters. If the string is a keyword, it adds the configured TPascalCodeGenerator.KeywordPrefix (1407) and TPascalCodeGenerator.KeywordSuffix (1407) or an ampersand (&) if ForceAmpersand is `True`.

See also: TPascalCodeGenerator.IsKeyWord (1403), TPascalCodeGenerator.KeywordPrefix (1407), TPascalCodeGenerator.KeywordSuffix (1407)

53.4.15 TPascalCodeGenerator.MakePascalString

Synopsis: Convert string to Pascal string literal

Declaration: `function MakePascalString(const S: string; AddQuotes: Boolean) : string`

Visibility: public

Description: `MakePascalString` converts a string to a proper Pascal string literal by escaping single quotes. All single quotes in the input string are doubled to create valid Pascal string syntax. Optionally, the result can be wrapped in quotes.

53.4.16 TPascalCodeGenerator.PrettyPrint

Synopsis: Format string with proper capitalization

Declaration: `function PrettyPrint(const S: string) : string`

Visibility: public

Description: `PrettyPrint` formats a string by capitalizing the first character and leaving the rest unchanged. If the input string is empty, it returns an empty string.

53.4.17 TPascalCodeGenerator.AddLn

Synopsis: Add line to source code with current indentation

Declaration:
```
procedure AddLn(const aLine: string)
procedure AddLn(const TheLines: Array of string)
procedure AddLn(const TheLines: TStrings)
procedure AddLn(const Fmt: string; Args: Array of const)
```

Visibility: public

Description: `AddLn` adds one or more lines to the generated source code with the current indentation level applied. It supports adding single lines, arrays of strings, string lists or formatted strings. Each line is prefixed with the current indentation.

See also: TPascalCodeGenerator.Indent (1402), TPascalCodeGenerator.Undent (1402), TPascalCodeGenerator.Source (1405)

53.4.18 TPascalCodeGenerator.Comment

Synopsis: Add comment to source code

Declaration:
```
procedure Comment(const AComment: string; Curly: Boolean)
procedure Comment(const AComment: Array of string)
procedure Comment(const AComment: TStrings)
```

Visibility: public

Description: `Comment` adds comments to the generated source code. It supports single line comments (using // or { }) and multi-line comments using curly braces. For single strings, the Curly parameter determines the comment style. For arrays of strings or string lists, multi-line curly brace comments are generated.

See also: TPascalCodeGenerator.AddLn (1404)

53.4.19 TPascalCodeGenerator.ClassHeader

Synopsis: Add class header comment (deprecated)

Declaration:
```
procedure ClassHeader(const AClassName: string);  Virtual
```

Visibility: public

Description: `ClassHeader` adds a class header comment. This method is deprecated in favor of TPascalCodeGenerator.ClassComment (1399).

See also: TPascalCodeGenerator.ClassComment (1399)

53.4.20 TPascalCodeGenerator.SimpleMethodBody

Synopsis: Generate simple method body

Declaration:
```
procedure SimpleMethodBody(Lines: Array of string);  Virtual
```

Visibility: public

Description: `SimpleMethodBody` generates a complete method body with begin/end block and proper indentation. It adds an empty line, then 'begin', indents and adds the provided lines, undents and adds 'end;', followed by another empty line.

See also: TPascalCodeGenerator.AddLn (1404), TPascalCodeGenerator.Indent (1402), TPascalCodeGenerator.Undent (1402)

53.4.21 TPascalCodeGenerator.SaveToStream

Synopsis: Save generated source code to stream

Declaration: `procedure SaveToStream(const AStream: TStream)`

Visibility: public

Description: `SaveToStream` saves the generated source code to the specified stream. The entire content of the TPascalCodeGenerator.Source (1405) string list is written to the stream.

See also: TPascalCodeGenerator.SaveToFile (1405), TPascalCodeGenerator.Source (1405)

53.4.22 TPascalCodeGenerator.SaveToFile

Synopsis: Save generated source code to file

Declaration: `procedure SaveToFile(const AFileName: string)`

Visibility: public

Description: `SaveToFile` saves the generated source code to the specified file. If no output unit name is set and the source is empty, it automatically sets the output unit name based on the filename.

See also: TPascalCodeGenerator.SaveToStream (1405), TPascalCodeGenerator.OutputUnitName (1406), TPascalCodeGenerator.Source (1405)

53.4.23 TPascalCodeGenerator.Source

Synopsis: Generated source code string list

Declaration: `Property Source : TStrings`

Visibility: public

Access: Read

Description: `Source` contains the generated source code as a read-only string list. Each line in the list represents one line of generated Pascal code. This property provides access to the complete generated output.

See also: TPascalCodeGenerator.AddLn (1404), TPascalCodeGenerator.SaveToFile (1405), TPascalCodeGenerator.SaveToStream (1405)

53.4.24 TPascalCodeGenerator.CurrentSection

Synopsis: Current code section type

Declaration: `Property CurrentSection : TCodeSection`

Visibility: public

Access: Read,Write

Description: `CurrentSection` indicates the current section type being generated. Reading this property returns the section type at the top of the section stack. Writing to this property changes the current section without affecting the stack.

See also: TCodeSection (1399), TPascalCodeGenerator.EnsureSection (1401), TPascalCodeGenerator.PushSection (1401), TPascalCodeGenerator.PopSection (1401)

53.4.25 TPascalCodeGenerator.OutputUnitName

Synopsis: Name of the output unit

Declaration: `Property OutputUnitName : string`

Visibility: published

Access: Read,Write

Description: `OutputUnitName` specifies the name of the Pascal unit being generated. This name is used in the unit clause and can be automatically derived from the filename when saving.

See also: TPascalCodeGenerator.CreateUnitClause (1402), TPascalCodeGenerator.SaveToFile (1405)

53.4.26 TPascalCodeGenerator.ExtraUnits

Synopsis: Additional units for uses clause

Declaration: `Property ExtraUnits : string`

Visibility: published

Access: Read,Write

Description: `ExtraUnits` contains additional units to include in the uses clause for non-dotted unit support. These units are added to the base units when generating the interface uses clause.

See also: TPascalCodeGenerator.DottedExtraUnits (1399), TPascalCodeGenerator.CreateHeader (1402)

53.4.27 TPascalCodeGenerator.LicenseText

Synopsis: License text for header comment

Declaration: `Property LicenseText : TStrings`

Visibility: published

Access: Read,Write

Description: `LicenseText` contains the license text that will be included as a comment at the beginning of the generated unit. This text is output by TPascalCodeGenerator.CreateHeader (1402).

See also: TPascalCodeGenerator.CreateHeader (1402)

53.4.28 TPascalCodeGenerator.Switches

Synopsis: Compiler switches for the unit

Declaration: `Property Switches : TStrings`

Visibility: published

Access: Read,Write

Description: `Switches` contains a list of compiler switches that will be included in the generated unit header. By default, it includes MODE ObjFPC and H+ switches.

See also: TPascalCodeGenerator.CreateHeader (1402)

53.4.29 TPascalCodeGenerator.OnLog

Synopsis: Event handler for log messages

Declaration: `Property OnLog : TCodeGeneratorLogEvent`

Visibility: published

Access: Read,Write

Description: `OnLog` is an event that is triggered when the code generator wants to log information or warnings. Assign an event handler to receive these messages for debugging or user feedback.

See also: TCodeGeneratorLogEvent (1398), TCodegenLogType (1398)

53.4.30 TPascalCodeGenerator.AddTimeStamp

Synopsis: Whether to include timestamp in header

Declaration: `Property AddTimeStamp : Boolean`

Visibility: published

Access: Read,Write

Description: `AddTimeStamp` determines whether a timestamp comment is added to the generated unit header. When `True`, TPascalCodeGenerator.CreateHeader (1402) includes a 'Generated on:' comment with the current date and time.

See also: TPascalCodeGenerator.CreateHeader (1402)

53.4.31 TPascalCodeGenerator.KeywordSuffix

Synopsis: Suffix for escaped keywords

Declaration: `Property KeywordSuffix : string`

Visibility: published

Access: Read,Write

Description: `KeywordSuffix` specifies the suffix string to append to escaped Pascal keywords. This is used by TPascalCodeGenerator.EscapeKeyWord (1403) when ForceAmpersand is `False`.

See also: TPascalCodeGenerator.EscapeKeyWord (1403), TPascalCodeGenerator.KeywordPrefix (1407)

53.4.32 TPascalCodeGenerator.KeywordPrefix

Synopsis: Prefix for escaped keywords

Declaration: `Property KeywordPrefix : string`

Visibility: published

Access: Read,Write

Description: `KeywordPrefix` specifies the prefix string to prepend to escaped Pascal keywords. By default, this is set to '&' and is used by TPascalCodeGenerator.EscapeKeyWord (1403) when ForceAmpersand is `False`.

See also: TPascalCodeGenerator.EscapeKeyWord (1403), TPascalCodeGenerator.KeywordSuffix (1407)

Chapter 54

Reference for unit 'Pipes'

54.1 Used units

Table 54.1: Used units by unit 'Pipes'

Name	Page
Classes	??
System	??
sysutils	??

54.2 Overview

The Pipes unit implements streams that are wrappers around the OS's pipe functionality. It creates a pair of streams, and what is written to one stream can be read from another.

54.3 Constants, types and variables

54.3.1 Constants

```
ENoSeekMsg = 'Cannot seek on pipes'
```

Constant used in EPipeSeek (1409) exception.

```
EPipeMsg = 'Failed to create pipe.'
```

Constant used in EPipeCreation (1409) exception.

54.4 Procedures and functions

54.4.1 CreatePipeHandles

Synopsis: Function to create a set of pipe handles.

Declaration:
```
function CreatePipeHandles(var Inhandle: THandle;
                           var OutHandle: THandle;
                           APipeBufferSize: Cardinal) : Boolean
```

Visibility: default

Description: `CreatePipeHandles` provides an OS-independent way to create a set of pipe filehandles. These handles are inheritable to child processes. The reading end of the pipe is returned in `InHandle`, the writing end in `OutHandle`.

Errors: On error, `False` is returned.

See also: CreatePipeStreams (1409)

54.4.2 CreatePipeStreams

Synopsis: Create a pair of pipe stream.

Declaration:
```
procedure CreatePipeStreams(var InPipe: TInputPipeStream;
                            var OutPipe: TOutputPipeStream)
```

Visibility: default

Description: `CreatePipeStreams` creates a set of pipe file descriptors with CreatePipeHandles (1408), and if that call is successful, a pair of streams is created: `InPipe` and `OutPipe`.

On some systems (notably: windows) the size of the buffer to be used for communication between 2 ends of the buffer can be specified in the APipeBufferSize (1408) parameter. This parameter is ignored on systems that do not support setting the buffer size.

Errors: If no pipe handles could be created, an EPipeCreation (1409) exception is raised.

See also: CreatePipeHandles (1408), TInputPipeStream (1410), TOutputPipeStream (1412)

54.5 EPipeCreation

54.5.1 Description

Exception raised when an error occurred during the creation of a pipe pair.

54.6 EPipeError

54.6.1 Description

Exception raised when an invalid operation is performed on a pipe stream.

54.7 EPipeSeek

54.7.1 Description

Exception raised when an invalid seek operation is attempted on a pipe.

54.8 TInputPipeStream

54.8.1 Description

`TInputPipeStream` is created by the CreatePipeStreams (1409) call to represent the reading end of a pipe. It is a TStream (**??**) descendent which does not allow writing, and which mimics the seek operation.

See also: TStream (**??**), CreatePipeStreams (1409), TOutputPipeStream (1412)

54.8.2 Method overview

Page	Method	Description
1410	Destroy	Destroy this instance of the input pipe stream.
1411	Read	Read data from the stream to a buffer.
1411	Seek	Set the current position of the stream.
1410	Write	Write data to the stream.

54.8.3 Property overview

Page	Properties	Access	Description
1411	NumBytesAvailable	r	Number of bytes available for reading.

54.8.4 TInputPipeStream.Destroy

Synopsis: Destroy this instance of the input pipe stream.

Declaration: `destructor Destroy;  Override`

Visibility: public

Description: `Destroy` overrides the destructor to close the pipe handle, prior to calling the inherited destructor.

See also: TInputPipeStream.Create (1410)

54.8.5 TInputPipeStream.Write

Synopsis: Write data to the stream.

Declaration: `function Write(const Buffer; Count: LongInt) : LongInt;  Override`

Visibility: public

Description: `Write` overrides the parent implementation of `Write`. On a `TInputPipeStream` will always raise an exception, as the pipe is read-only.

Errors: An EStreamError (**??**) exception is raised when this function is called.

See also: Read (1411), Seek (1411)

54.8.6 TInputPipeStream.Seek

Synopsis: Set the current position of the stream.

Declaration:
```
function Seek(const Offset: Int64; Origin: TSeekOrigin) : Int64
                  ;  Override
```

Visibility: public

Description: `Seek` overrides the standard `Seek` implementation. Normally, pipe streams stderr are not seekable. The `TInputPipeStream` stream tries to provide seek capabilities for the following limited number of cases:

Origin=soFromBeginning If `Offset` is larger than the current position, then the remaining bytes are skipped by reading them from the stream and discarding them.

Origin=soFromCurrent If `Offset` is zero, the current position is returned. If it is positive, then `Offset` bytes are skipped by reading them from the stream and discarding them, if the stream is of type `iosInput`.

All other cases will result in a `EPipeSeek` exception.

Errors: An EPipeSeek (1409) exception is raised if the stream does not allow the requested seek operation.

See also: EPipeSeek (1409), Seek (**??**)

54.8.7 TInputPipeStream.Read

Synopsis: Read data from the stream to a buffer.

Declaration:
```
function Read(var Buffer; Count: LongInt) : LongInt;  Override
```

Visibility: public

Description: `Read` calls the inherited read and adjusts the internal position pointer of the stream.

Errors: None.

See also: Write (1410), Seek (1411)

54.8.8 TInputPipeStream.NumBytesAvailable

Synopsis: Number of bytes available for reading.

Declaration:
```
Property NumBytesAvailable : DWord
```

Visibility: public

Access: Read

Description: `NumBytesAvailable` is the number of bytes available for reading. This is the number of bytes in the OS buffer for the pipe. It is not a number of bytes in an internal buffer.

If this number is nonzero, then reading `NumBytesAvailable` bytes from the stream will not block the process. Reading more than `NumBytesAvailable` bytes will block the process, while it waits for the requested number of bytes to become available.

See also: TInputPipeStream.Read (1411)

54.9 TOutputPipeStream

54.9.1 Description

`TOutputPipeStream` is created by the CreatePipeStreams (1409) call to represent the writing end of a pipe. It is a TStream (**??**) descendent which does not allow reading.

See also: TStream (**??**), CreatePipeStreams (1409), TInputPipeStream (1410)

54.9.2 Method overview

Page	Method	Description
1412	Destroy	Destroy this instance of the output pipe stream.
1412	Read	Read data from the stream.
1412	Seek	Sets the position in the stream.

54.9.3 Property overview

Page	Properties	Access	Description
1413	DontClose	rw	

54.9.4 TOutputPipeStream.Destroy

Synopsis: Destroy this instance of the output pipe stream.

Declaration:
```
destructor Destroy;  Override
```

Visibility: public

Description: `Destroy` overrides the destructor to close the pipe handle, prior to calling the inherited destructor.

See also: THandleStream.Create (**??**)

54.9.5 TOutputPipeStream.Seek

Synopsis: Sets the position in the stream.

Declaration:
```
function Seek(const Offset: Int64; Origin: TSeekOrigin) : Int64
                  ;  Override
```

Visibility: public

Description: `Seek` is overridden in `TOutputPipeStream`. Calling this method will always raise an exception: an output pipe is not seekable.

Errors: An EPipeSeek (1409) exception is raised if this method is called.

54.9.6 TOutputPipeStream.Read

Synopsis: Read data from the stream.

Declaration:
```
function Read(var Buffer; Count: LongInt) : LongInt;  Override
```

Visibility: public

Description: `Read` overrides the parent `Read` implementation. It always raises an exception, because a output pipe is write-only.

Errors: An EStreamError (**??**) exception is raised when this function is called.

See also: Seek (1412)

54.9.7 TOutputPipeStream.DontClose

Declaration: `Property DontClose : Boolean`

Visibility: public

Access: Read,Write

Chapter 55

Reference for unit 'pooledmm'

55.1 Used units

Table 55.1: Used units by unit 'pooledmm'

Name	Page
Classes	??
System	??

55.2 Overview

pooledmm is a memory manager class which uses pools of blocks. Since it is a higher-level implementation of a memory manager which works on top of the FPC memory manager, It also offers more debugging and analysis tools. It is used mainly in the LCL and Lazarus IDE.

55.3 Constants, types and variables

55.3.1 Types

```
PPooledMemManagerItem = ^TPooledMemManagerItem
```

`PPooledMemManagerItem` is a pointer type, pointing to a TPooledMemManagerItem (1415) item, used in a linked list.

```
TEnumItemsMethod = procedure(Item: Pointer) of object
```

`TEnumItemsMethod` is a prototype for the callback used in the TNonFreePooledMemManager.EnumerateItems (1416) call. The parameter `Item` will be set to each of the pointers in the item list of TNonFreePooledMemManager (1415).

55.4 TPooledMemManagerItem

```
TPooledMemManagerItem = record
```

```
  Next : PPooledMemManagerItem;
end
```

TPooledMemManagerItem is used internally by the TPooledMemManager (1417) class to maintain the free list block. It simply points to the next free block.

55.5 TNonFreePooledMemManager

55.5.1 Description

TNonFreePooledMemManager keeps a list of fixed-size memory blocks in memory. Each block has the same size, making it suitable for storing a lot of records of the same type. It does not free the items stored in it, except when the list is cleared as a whole.

It allocates memory for the blocks in a exponential way, i.e. each time a new block of memory must be allocated, it's size is the double of the last block. The first block will contain 8 items.

55.5.2 Method overview

Page	Method	Description
1415	Clear	Clears the memory.
1415	Create	Creates a new instance of TNonFreePooledMemManager.
1416	Destroy	Removes the TNonFreePooledMemManager instance from memory.
1416	EnumerateItems	Enumerate all items in the list.
1416	NewItem	Return a pointer to a new memory block.

55.5.3 Property overview

Page	Properties	Access	Description
1416	ItemSize	r	Size of an item in the list.

55.5.4 TNonFreePooledMemManager.Clear

Synopsis: Clears the memory.

Declaration: `procedure Clear`

Visibility: public

Description: Clear clears all blocks from memory, freeing the allocated memory blocks. None of the pointers returned by NewItem (1416) is valid after a call to Clear

See also: NewItem (1416)

55.5.5 TNonFreePooledMemManager.Create

Synopsis: Creates a new instance of TNonFreePooledMemManager.

Declaration: `constructor Create(TheItemSize: Integer)`

Visibility: public

Description: `Create` creates a new instance of `TNonFreePooledMemManager` and sets the item size to `TheItemSize`.

Errors: If not enough memory is available, an exception may be raised.

See also: TNonFreePooledMemManager.ItemSize (1416)

55.5.6 TNonFreePooledMemManager.Destroy

Synopsis: Removes the `TNonFreePooledMemManager` instance from memory.

Declaration: `destructor Destroy;  Override`

Visibility: public

Description: `Destroy` clears the list, clears the internal structures, and then calls the inherited `Destroy`.

`Destroy` should never be called directly. Instead `Free` should be used, or `FreeAndNil`

See also: TNonFreePooledMemManager.Create (1415), TNonFreePooledMemManager.Clear (1415)

55.5.7 TNonFreePooledMemManager.NewItem

Synopsis: Return a pointer to a new memory block.

Declaration: `function NewItem : Pointer`

Visibility: public

Description: `NewItem` returns a pointer to an unused memory block of size ItemSize (1416). It will allocate new memory on the heap if necessary.

Note that there is no way to mark the memory block as free, except by clearing the whole list.

Errors: If no more memory is available, an exception may be raised.

See also: TNonFreePooledMemManager.Clear (1415)

55.5.8 TNonFreePooledMemManager.EnumerateItems

Synopsis: Enumerate all items in the list.

Declaration: `procedure EnumerateItems(const Method: TEnumItemsMethod)`

Visibility: public

Description: `EnumerateItems` will enumerate over all items in the list, passing the items to `Method`. This can be used to execute certain operations on all items in the list. (for example, simply list them)

55.5.9 TNonFreePooledMemManager.ItemSize

Synopsis: Size of an item in the list.

Declaration: `Property ItemSize : Integer`

Visibility: public

Access: Read

Description: `ItemSize` is the size of a single block in the list. It's a fixed size determined when the list is created.

See also: TNonFreePooledMemManager.Create (1415)

55.6 TPooledMemManager

55.6.1 Description

`TPooledMemManager` is a class which maintains a linked list of blocks, represented by the TPooled-MemManagerItem (1415) record. It should not be used directly, but should be descended from and the descendent should implement the actual memory manager.

See also: TPooledMemManagerItem (1415)

55.6.2 Method overview

Page	Method	Description
1417	Clear	Clears the list.
1417	Create	Creates a new instance of the `TPooledMemManager` class.
1418	Destroy	Removes an instance of `TPooledMemManager` class from memory.

55.6.3 Property overview

Page	Properties	Access	Description
1419	AllocatedCount	r	Total number of allocated items in the list.
1418	Count	r	Number of items in the list.
1419	FreeCount	r	Number of free items in the list.
1419	FreedCount	r	Total number of freed items in the list.
1418	MaximumFreeCountRatio	rw	Maximum ratio of free items over total items.
1418	MinimumFreeCount	rw	Minimum count of free items in the list.

55.6.4 TPooledMemManager.Clear

Synopsis: Clears the list.

Declaration: `procedure Clear`

Visibility: public

Description: `Clear` clears the list, it disposes all items in the list.

See also: TPooledMemManager.FreedCount (1419)

55.6.5 TPooledMemManager.Create

Synopsis: Creates a new instance of the `TPooledMemManager` class.

Declaration: `constructor Create`

Visibility: public

Description: `Create` initializes all necessary properties and then calls the inherited create.

See also: TPooledMemManager.Destroy (1418)

55.6.6 TPooledMemManager.Destroy

Synopsis: Removes an instance of `TPooledMemManager` class from memory.

Declaration: `destructor Destroy;  Override`

Visibility: public

Description: `Destroy` calls Clear (1417) and then calls the inherited destroy.

`Destroy` should never be called directly. Instead `Free` should be used, or `FreeAndNil`

See also: TPooledMemManager.Create (1417)

55.6.7 TPooledMemManager.MinimumFreeCount

Synopsis: Minimum count of free items in the list.

Declaration: `Property MinimumFreeCount : Integer`

Visibility: public

Access: Read,Write

Description: `MinimumFreeCount` is the minimum number of free items in the linked list. When disposing an item in the list, the number of items is checked, and only if the required number of free items is present, the item is actually freed.

The default value is 100000

See also: TPooledMemManager.MaximumFreeCountRatio (1418)

55.6.8 TPooledMemManager.MaximumFreeCountRatio

Synopsis: Maximum ratio of free items over total items.

Declaration: `Property MaximumFreeCountRatio : Integer`

Visibility: public

Access: Read,Write

Description: `MaximumFreeCountRatio` is the maximum ratio (divided by 8) of free elements over the total amount of elements: When disposing an item in the list, if the number of free items is higher than this ratio, the item is freed.

The default value is 8.

See also: TPooledMemManager.MinimumFreeCount (1418)

55.6.9 TPooledMemManager.Count

Synopsis: Number of items in the list.

Declaration: `Property Count : Integer`

Visibility: public

Access: Read

Description: `Count` is the total number of items allocated from the list.

See also: TPooledMemManager.FreeCount (1419), TPooledMemManager.AllocatedCount (1419), TPooled-MemManager.FreedCount (1419)

55.6.10 TPooledMemManager.FreeCount

Synopsis: Number of free items in the list.

Declaration: `Property FreeCount : Integer`

Visibility: public

Access: Read

Description: `FreeCount` is the current total number of free items in the list.

See also: TPooledMemManager.Count (1418), TPooledMemManager.AllocatedCount (1419), TPooledMemManager.FreedCount (1419)

55.6.11 TPooledMemManager.AllocatedCount

Synopsis: Total number of allocated items in the list.

Declaration: `Property AllocatedCount : Int64`

Visibility: public

Access: Read

Description: `AllocatedCount` is the total number of newly allocated items on the list.

See also: TPooledMemManager.Count (1418), TPooledMemManager.FreeCount (1419), TPooledMemManager.FreedCount (1419)

55.6.12 TPooledMemManager.FreedCount

Synopsis: Total number of freed items in the list.

Declaration: `Property FreedCount : Int64`

Visibility: public

Access: Read

Description: `FreedCount` is the total number of elements actually freed in the list.

See also: TPooledMemManager.Count (1418), TPooledMemManager.FreeCount (1419), TPooledMemManager.AllocatedCount (1419)

Chapter 56

Reference for unit 'process'

56.1 Used units

Table 56.1: Used units by unit 'process'

Name	Page
Classes	??
Math	??
Pipes	1408
System	??
sysutils	??

56.2 Overview

The Process unit contains the code for the TProcess (1426) component, a cross-platform component to start and control other programs, offering also access to standard input and output for these programs.

`TProcess` does not handle wildcard expansion, does not support complex pipelines as in Unix. If this behaviour is desired, the shell can be executed with the pipeline as the command it should execute.

56.3 Constants, types and variables

56.3.1 Types

```
TOnRunCommandEvent = procedure(Sender: TObject; Context: TObject;
  Status: TRunCommandEventCode;
  const Message: string) of object
```

`TOnRunCommandEvent` is the event handler prototype for the various events emitted by the TProcess (1426) class during the RunCommandLoop (1431) call.

```
TprocessChar = Char
```

TProcessChar is a single-byte character in the single-byte version of TProcess, but is a 2-byte character in the unicode version of TProcess.

```
TProcessClass = Class of TPROCESS
```

Class of TProcess.

```
TProcessForkEvent = procedure(Sender: TObject) of object
```

TProcessForkEvent is the prototype for TProcess.OnForkEvent (1436). It is a simple procedure, as the idea is that only process-global things should be performed in this event handler.

```
TProcessOption = (poRunSuspended,poWaitOnExit,poUsePipes,
  poStderrToOutPut,poNoConsole,poNewConsole,
  poDefaultErrorMode,poNewProcessGroup,poDebugProcess,
  poDebugOnlyThisProcess,poDetached,poPassInput,
  poRunIdle)
```

Table 56.2: Enumeration values for type TProcessOption

Value	Explanation
poDebugOnlyThisProcess	Do not follow processes started by this process (Win32 only).
poDebugProcess	Allow debugging of the process (Win32 only).
poDefaultErrorMode	Use default error handling.
poDetached	Runs a process using the DETACHED_PROCESS creation flag on Windows.
poNewConsole	Start a new console window for the process (Win32 only).
poNewProcessGroup	Start the process in a new process group (Win32 only).
poNoConsole	Do not allow access to the console window for the process (Win32 only).
poPassInput	Pass standard input handle on to new process.
poRunIdle	Signals an event handler to wait for output in the run loop for a process.
poRunSuspended	Start the process in suspended state.
poStderrToOutPut	Redirect standard error to the standard output stream.
poUsePipes	Use pipes to redirect standard input and output.
poWaitOnExit	Wait for the process to terminate before returning.

When a new process is started using TProcess.Execute (1428), these options control the way the process is started. Note that not all options are supported on all platforms.

```
TProcessOptions = Set of TProcessOption
```

Set of TProcessOption (1421).

```
TProcessPriority = (ppHigh,ppIdle,ppNormal,ppRealTime,ppBelowNormal
  ,
                    ppAboveNormal)
```

Table 56.3: Enumeration values for type TProcessPriority

Value	Explanation
ppAboveNormal	Above normal priority.
ppBelowNormal	Below normal priority.
ppHigh	The process runs at higher than normal priority.
ppIdle	The process only runs when the system is idle (i.e. has nothing else to do).
ppNormal	The process runs at normal priority.
ppRealTime	The process runs at real-time priority.

This enumerated type determines the priority of the newly started process. It translates to default platform specific constants. If finer control is needed, then platform-dependent mechanism need to be used to set the priority.

```
TProcessString = String
```

`TProcessString` is a single-byte string in the single-byte version of TProcess, but is a 2-byte (unicode) string in the unicode version of `TProcess`.

```
TProcessStringList = TStringList
```

`TProcessStringList` is an alias for TProcessStrings (1422) in unicode code, or an alias for the TStringList (**??**) class in single-byte string mode.

```
TProcessStrings = TStrings
```

`TProcessStrings` is a simple string list class which, depending on the version (unicode or not) contains unicode strings or single-byte strings: in the latter case it is an alias for the #rtl.classes.TStrings (**??**) class.

```
TRunCommandEventCode = (RunCommandIdle,RunCommandReadOutputString
  ,
                        RunCommandReadOutputStream,RunCommandFinished
  ,
                        RunCommandException)
```

Table 56.4: Enumeration values for type TRunCommandEventCode

Value	Explanation
RunCommandException	An error happened during reading of the command.
RunCommandFinished	The command finished.
RunCommandIdle	No data was available for reading.
RunCommandReadOutputStream	Output from the command was read.
RunCommandReadOutputString	Output from the command was read as a string.

`TRunCommandEventCode` is an enumerated type indicating the stage at which a process is during the RunCommandLoop (1431) call, reported through the TProcess.OnRunCommandEvent (1435) event handler.

```
TRunCommandEventCodeSet = Set of TRunCommandEventCode
```

TRunCommandEventCodeSet is a set of TRunCommandEventCode (1422) values.

```
TShowWindowOptions = (swoNone,swoHIDE,swoMaximize,swoMinimize,
  swoRestore,swoShow,swoShowDefault,
  swoShowMaximized,swoShowMinimized,
  swoshowMinNOActive,swoShowNA,swoShowNoActivate,
  swoShowNormal)
```

Table 56.5: Enumeration values for type TShowWindowOptions

Value	Explanation
swoHIDE	The main window is hidden.
swoMaximize	The main window is maximized.
swoMinimize	The main window is minimized.
swoNone	Allow system to position the window.
swoRestore	Restore the previous position.
swoShow	Show the main window.
swoShowDefault	When showing Show the main window on.
swoShowMaximized	The main window is shown maximized.
swoShowMinimized	The main window is shown minimized.
swoshowMinNOActive	The main window is shown minimized but not activated.
swoShowNA	The main window is shown but not activated.
swoShowNoActivate	The main window is shown but not activated.
swoShowNormal	The main window is shown normally.

This type describes what the new process' main window should look like. Most of these have only effect on Windows. They are ignored on other systems.

```
TStartupOption = (suoUseShowWindow,suoUseSize,suoUsePosition,
  suoUseCountChars,suoUseFillAttribute)
```

Table 56.6: Enumeration values for type TStartupOption

Value	Explanation
suoUseCountChars	Use the console character width as specified in TProcess (1426).
suoUseFillAttribute	Use the console fill attribute as specified in TProcess (1426).
suoUsePosition	Use the window sizes as specified in TProcess (1426).
suoUseShowWindow	Use the Show Window options specified in TShowWindowOption (1423).
suoUseSize	Use the window sizes as specified in TProcess (1426).

These options are mainly for Win32, and determine what should be done with the application once it's started.

```
TStartupOptions = Set of TStartupOption
```

Set of TStartUpOption (1423).

56.3.2 Variables

```
DefaultTProcess : TProcessClass = TPROCESS
```

`DefaultTProcess` is the process class used by the RunCommand (1425) and RunCommandInDir (1425) calls. You can set it to customize the process class to use during these calls. By default the `TProcess` class is used.

```
TryTerminals : Array of string
```

`TryTerminals` is used under UNIX to test for available terminal programs in the DetectXTerm (1424) function. If XTermProgram (1424) is empty, each item in this list will be searched in the path, and used as a terminal program if it was found.

```
XTermProgram : string
```

`XTermProgram` is the terminal program that is used. If empty, it will be set the first time DetectXTerm (1424) is called.

56.4 Procedures and functions

56.4.1 CommandToList

Synopsis: Convert a command-line to a list of command options.

Declaration: `procedure CommandToList(S: TProcessString; List: TProcessStrings)`

Visibility: default

Description: `CommandToList` splits the string `S` in command-line arguments that are returned, one per item, in the `List` stringlist. Command-line arguments are separated by whitespace (space, tab, CR and LF characters). If an argument needs to contain a space character, it can be surrounded in quote characters (single or double quotes).

Errors: There is currently no way to specify a quote character inside a quoted argument.

See also: TProcess.CommandLine (1437)

56.4.2 DetectXTerm

Synopsis: Detect the terminal program.

Declaration: `function DetectXTerm : string`

Visibility: default

Description: `DetectXTerm` checks if XTermProgram (1424) is set. if so, it returns that. If `XTermProgram` is empty, the list specified in TryTerminals (1424) is tested for existence. If none is found, then the `DESKTOP_SESSION` environment variable is examined:

kde konsole is used if it is found.

gnome gnome-terminal is used if it is found

windowmaker aterm or xterm are used if found.

If after all this, no terminal is found, then a list of default programs is tested: 'x-terminal-emulator','xterm','aterm','wterm','rxvt

If a terminal program is found, then it is saved in `XTermProgram`, so the next call to `DetectXTerm` will re-use the value. If the search must be performed again, it is sufficient to set `XTermProgram` to the empty string.

See also: XTermProgram (1424), TryTerminals (1424), TProcess.XTermProgram (1444)

56.4.3 RunCommand

Synopsis: Execute a command in the current working directory.

Declaration:
```
function RunCommand(const exename: TProcessString;
                    const commands: Array of TProcessString;
                    out outputstring: string; Options: TProcessOptions;
                    SWOptions: TShowWindowOptions) : Boolean
function RunCommand(const cmdline: TProcessString;
                    out outputstring: string) : Boolean
```

Visibility: default

Description: `RunCommand` runs RunCommandInDir (1425) with an empty current working directory.

The version using `CmdLine` attempts to split the command line in a binary and separate command-line arguments. This version of the function is deprecated.

See also: RunCommandInDir (1425)

56.4.4 RunCommandIndir

Synopsis: Run a command in a specific directory.

Declaration:
```
function RunCommandIndir(const curdir: TProcessString;
                         const exename: TProcessString;
                         const commands: Array of TProcessString;
                         out outputstring: string;
                         out exitstatus: Integer;
                         Options: TProcessOptions;
                         SWOptions: TShowWindowOptions) : Integer
function RunCommandIndir(const curdir: TProcessString;
                         const exename: TProcessString;
                         const commands: Array of TProcessString;
                         out outputstring: string;
                         Options: TProcessOptions;
                         SWOptions: TShowWindowOptions) : Boolean
function RunCommandInDir(const curdir: TProcessString;
                         const cmdline: TProcessString;
                         out outputstring: string) : Boolean
```

Visibility: default

Description: `RunCommandInDir` will execute binary `exename` with command-line options `commands`, setting `curdir` as the current working directory for the command. The Options (1421) are taken into consideration (`poRunSuspended`,`poWaitOnExit` are removed from the set). The output of the command is captured, and returned in the string `OutputString`. The function waits for the command to finish, and returns `True` if the command was started successfully, `False` otherwise. In the case where the return value is an integer, it is zero for success, and -1 on error.

If a `ExitStatus` parameter is specified the exit status of the command is returned in this parameter.

The version using `cmdline` attempts to split the command line in a binary and separate command-line arguments. This version of the function is deprecated.

Errors: On error, `False` is returned.

See also: TProcess (1426), RunCommand (1425), TProcessOptions (1421)

56.5 EProcess

56.5.1 Description

Exception raised when an error occurs in a TProcess routine.

See also: TProcess (1426)

56.6 TPROCESS

56.6.1 Description

`TProcess` is a component that can be used to start and control other processes (programs/binaries). It contains a lot of options that control how the process is started. Many of these are Win32 specific, and have no effect on other platforms, so they should be used with care.

The simplest way to use this component is to create an instance, set the CommandLine (1437) property to the full pathname of the program that should be executed, and call Execute (1428). To determine whether the process is still running (i.e. has not stopped executing), the Running (1441) property can be checked.

More advanced techniques can be used with the Options (1440) settings.

See also: Create (1428), Execute (1428), Running (1441), CommandLine (1437), Options (1440)

56.6.2 Method overview

Page	Method	Description
1429	CloseInput	Close the input stream of the process.
1429	CloseOutput	Close the output stream of the process.
1429	CloseStderr	Close the error stream of the process.
1428	Create	Create a new instance of the `TProcess` class.
1428	Destroy	Destroy this instance of `TProcess`.
1428	Execute	Execute the program with the given options.
1431	ReadInputStream	Read available data from input stream.
1429	Resume	Resume execution of a suspended process.
1431	RunCommandLoop	Execute command and collect output in strings.
1430	Suspend	Suspend a running process.
1430	Terminate	Terminate a running process.
1430	WaitOnExit	Wait for the program to stop executing.

56.6.3 Property overview

Page	Properties	Access	Description
1436	Active	rw	Start or stop the process.
1436	ApplicationName	rw	Name of the application to start (deprecated).
1437	CommandLine	rw	Command-line to execute (deprecated).
1438	ConsoleTitle	rw	Title of the console window.
1439	CurrentDirectory	rw	Working directory of the process.
1439	Desktop	rw	Desktop on which to start the process.
1439	Environment	rw	Environment variables for the new process.
1437	Executable	rw	Executable name. Supersedes `CommandLine` and `ApplicationName`.
1435	ExitCode	r	Exit code of the process.
1434	ExitStatus	r	Exit status of the process.
1444	FillAttribute	rw	Color attributes of the characters in the console window (Windows only).
1432	Handle	r	Handle of the process.
1435	InheritHandles	rw	Should the created process inherit the open handles of the current process.
1433	Input	r	Stream connected to standard input of the process.
1436	OnForkEvent	rw	Event triggered after fork occurred on Linux.
1435	OnRunCommandEvent	rw	Event handler, called when `RunCommandLoop` is executing.
1440	Options	rw	Options to be used when starting the process.
1433	Output	r	Stream connected to standard output of the process.
1438	Parameters	rw	Command-line arguments. Supersedes `CommandLine`.
1436	PipeBufferSize	rw	Buffer size to be used when using pipes.
1440	Priority	rw	Priority at which the process is running.
1432	ProcessHandle	r	Alias for Handle (1432).
1432	ProcessID	r	ID of the process.
1435	RunCommandSleepTime	rw	Sleep time between attempts to collect data.
1441	Running	r	Determines whether the process is still running.
1442	ShowWindow	rw	Determines how the process main window is shown (Windows only).
1441	StartupOptions	rw	Additional (Windows) startup options.
1434	Stderr	r	Stream connected to standard diagnostic output of the process.
1432	ThreadHandle	r	Main process thread handle.
1433	ThreadID	r	ID of the main process thread.
1442	WindowColumns	rw	Number of columns in console window (windows only).
1442	WindowHeight	rw	Height of the process main window.
1443	WindowLeft	rw	X-coordinate of the initial window (Windows only).
1431	WindowRect	rw	Positions for the main program window.
1443	WindowRows	rw	Number of rows in console window (Windows only).
1443	WindowTop	rw	Y-coordinate of the initial window (Windows only).
1444	WindowWidth	rw	Height of the process main window (Windows only).
1444	XTermProgram	rw	XTerm program to use (UNIX only).

56.6.4 TPROCESS.Create

Synopsis: Create a new instance of the `TProcess` class.

Declaration: `constructor Create(AOwner: TComponent); Override`

Visibility: public

Description: `Create` creates a new instance of the `TProcess` class. After calling the inherited constructor, it simply sets some default values.

56.6.5 TPROCESS.Destroy

Synopsis: Destroy this instance of `TProcess`.

Declaration: `destructor Destroy; Override`

Visibility: public

Description: `Destroy` cleans up this instance of `TProcess`. Prior to calling the inherited destructor, it cleans up any streams that may have been created. If a process was started and is still executed, it is *not* stopped, but the standard input/output/stderr streams are no longer available, because they have been destroyed.

Errors: None.

See also: Create (1428)

56.6.6 TPROCESS.Execute

Synopsis: Execute the program with the given options.

Declaration: `procedure Execute; Virtual`

Visibility: public

Description: `Execute` actually executes the program as specified in CommandLine (1437), applying as much as of the specified options as supported on the current platform.

If the `poWaitOnExit` option is specified in Options (1440), then the call will only return when the program has finished executing (or if an error occurred). If this option is not given, the call returns immediately, but the WaitOnExit (1430) call can be used to wait for it to close, or the Running (1441) call can be used to check whether it is still running.

The TProcess.Terminate (1430) call can be used to terminate the program if it is still running, or the Suspend (1430) call can be used to temporarily stop the program's execution.

The ExitStatus (1434) function can be used to check the program's exit status, after it has stopped executing.

Errors: On error a EProcess (1426) exception is raised.

See also: TProcess.Running (1441), TProcess.WaitOnExit (1430), TProcess.Terminate (1430), TProcess.Suspend (1430), TProcess.Resume (1429), TProcess.ExitStatus (1434), TProcess.ExitCode (1435)

56.6.7 TPROCESS.CloseInput

Synopsis: Close the input stream of the process.

Declaration: `procedure CloseInput;  Virtual`

Visibility: public

Description: `CloseInput` closes the input file descriptor of the process, that is, it closes the handle of the pipe to standard input of the process.

See also: Input (1433), StdErr (1434), Output (1433), CloseOutput (1429), CloseStdErr (1429)

56.6.8 TPROCESS.CloseOutput

Synopsis: Close the output stream of the process.

Declaration: `procedure CloseOutput;  Virtual`

Visibility: public

Description: `CloseOutput` closes the output file descriptor of the process, that is, it closes the handle of the pipe to standard output of the process.

See also: Output (1433), Input (1433), StdErr (1434), CloseInput (1429), CloseStdErr (1429)

56.6.9 TPROCESS.CloseStderr

Synopsis: Close the error stream of the process.

Declaration: `procedure CloseStderr;  Virtual`

Visibility: public

Description: `CloseStdErr` closes the standard error file descriptor of the process, that is, it closes the handle of the pipe to standard error output of the process.

See also: Output (1433), Input (1433), StdErr (1434), CloseInput (1429), CloseStdErr (1429)

56.6.10 TPROCESS.Resume

Synopsis: Resume execution of a suspended process.

Declaration: `function Resume : Integer;  Virtual`

Visibility: public

Description: `Resume` should be used to let a suspended process resume it's execution. It should be called in particular when the `poRunSuspended` flag is set in Options (1440).

Errors: None.

See also: TProcess.Suspend (1430), TProcess.Options (1440), TProcess.Execute (1428), TProcess.Terminate (1430)

56.6.11 TPROCESS.Suspend

Synopsis: Suspend a running process.

Declaration:
```
function Suspend : Integer;  Virtual
```

Visibility: public

Description: `Suspend` suspends a running process. If the call is successful, the process is suspended: it stops running, but can be made to execute again using the Resume (1429) call.

`Suspend` is fundamentally different from TProcess.Terminate (1430) which actually stops the process.

Errors: On error, a nonzero result is returned.

See also: TProcess.Options (1440), TProcess.Resume (1429), TProcess.Terminate (1430), TProcess.Execute (1428)

56.6.12 TPROCESS.Terminate

Synopsis: Terminate a running process.

Declaration:
```
function Terminate(AExitCode: Integer) : Boolean;  Virtual
```

Visibility: public

Description: `Terminate` stops the execution of the running program. It effectively stops the program.

On Windows, the program will report an exit code of `AExitCode`, on other systems, this value is ignored.

Errors: On error, a nonzero value is returned.

See also: TProcess.ExitStatus (1434), TProcess.Suspend (1430), TProcess.Execute (1428), TProcess.WaitOnExit (1430), TProcess.ExitCode (1435)

56.6.13 TPROCESS.WaitOnExit

Synopsis: Wait for the program to stop executing.

Declaration:
```
function WaitOnExit : Boolean
function WaitOnExit(Timeout: DWord) : Boolean
```

Visibility: public

Description: `WaitOnExit` waits for the running program to exit. It returns `True` if the wait was successful, or `False` if there was some error waiting for the program to exit.

Note that the return value of this function has changed. The old return value was a DWord with a platform dependent error code. To make things consistent and cross-platform, a boolean return type was used.

The `TimeOut` argument can be used to specify a timeout in milliseconds. If omitted, the call will wait indefinitely.

Errors: On error, `False` is returned. No extended error information is available, as it is highly system dependent.

See also: TProcess.ExitStatus (1434), TProcess.Terminate (1430), TProcess.Running (1441), TProcess.ExitCode (1435)

56.6.14 TPROCESS.ReadInputStream

Synopsis: Read available data from input stream.

Declaration:
```
function ReadInputStream(p: TInputPipeStream; var BytesRead: Integer;
                         var DataLength: Integer; var Data: string;
                         MaxLoops: Integer) : Boolean;  Virtual
function ReadInputStream(p: TInputPipeStream; data: TStream;
                         MaxLoops: Integer) : Boolean;  Virtual
```

Visibility: public

Description: `ReadInputStream` reads data from the given input pipe stream `p` after checking that data is available. It returns `True` if data was succesfully read from the file handle. In the variant with a string data, the data is placed in the string `Data`, and `DataLength` is updated with the new length, `BytesRead` is updated with the amount of bytes read. `MaxLoop` determines how often an attempt at reading data is made.

In the variant with a stream, the available data is simply written to the stream.

Errors: None.

56.6.15 TPROCESS.RunCommandLoop

Synopsis: Execute command and collect output in strings.

Declaration:
```
function RunCommandLoop(out outputstring: string;
                        out stderrstring: string;
                        out anexitstatus: Integer) : Integer;  Virtual
```

Visibility: public

Description: `RunCommandLoop` executes the command, and runs a loop to read output of the command: the output of the command is returned in the `outputstring` parameter, and the error output is returned in the `stderrstring` string.

During collection of data or on error, the TProcess.OnRunCommandEvent (1435) event handler is called during the various stages of the call. If it is not explicitly set, a sleep period specified by TProcess.RunCommandSleepTime (1435) is interjected between the various read calls.

The return value of this call is 1 for error, zero for success.

See also: TProcess.OnRunCommandEvent (1435), TProcess.RunCommandSleepTime (1435)

56.6.16 TPROCESS.WindowRect

Synopsis: Positions for the main program window.

Declaration:
```
Property WindowRect : Trect
```

Visibility: public

Access: Read,Write

Description: `WindowRect` can be used to specify the position of

56.6.17 TPROCESS.Handle

Synopsis: Handle of the process.

Declaration: `Property Handle : THandle`

Visibility: public

Access: Read

Description: `Handle` identifies the process. In Unix systems, this is the process ID. On windows, this is the process handle. It can be used to signal the process.

The handle is only valid after TProcess.Execute (1428) has been called. It is not reset after the process stopped.

See also: TProcess.ThreadHandle (1432), TProcess.ProcessID (1432), TProcess.ThreadID (1433)

56.6.18 TPROCESS.ProcessHandle

Synopsis: Alias for Handle (1432).

Declaration: `Property ProcessHandle : THandle`

Visibility: public

Access: Read

Description: `ProcessHandle` equals Handle (1432) and is provided for completeness only.

See also: TProcess.Handle (1432), TProcess.ThreadHandle (1432), TProcess.ProcessID (1432), TProcess.ThreadID (1433)

56.6.19 TPROCESS.ThreadHandle

Synopsis: Main process thread handle.

Declaration: `Property ThreadHandle : THandle`

Visibility: public

Access: Read

Description: `ThreadHandle` is the main process thread handle. On Unix, this is the same as the process ID, on Windows, this may be a different handle than the process handle.

The handle is only valid after TProcess.Execute (1428) has been called. It is not reset after the process stopped.

See also: TProcess.Handle (1432), TProcess.ProcessID (1432), TProcess.ThreadID (1433)

56.6.20 TPROCESS.ProcessID

Synopsis: ID of the process.

Declaration: `Property ProcessID : Integer`

Visibility: public

Access: Read

Description: `ProcessID` is the ID of the process. It is the same as the handle of the process on Unix systems, but on Windows it is different from the process Handle.

The ID is only valid after TProcess.Execute (1428) has been called. It is not reset after the process stopped.

See also: TProcess.Handle (1432), TProcess.ThreadHandle (1432), TProcess.ThreadID (1433)

56.6.21 TPROCESS.ThreadID

Synopsis: ID of the main process thread.

Declaration: `Property ThreadID : Integer`

Visibility: public

Access: Read

Description: `ProcessID` is the ID of the main process thread. It is the same as the handle of the main process thread (or the process itself) on Unix systems, but on Windows it is different from the thread Handle.

The ID is only valid after TProcess.Execute (1428) has been called. It is not reset after the process stopped.

See also: TProcess.ProcessID (1432), TProcess.Handle (1432), TProcess.ThreadHandle (1432)

56.6.22 TPROCESS.Input

Synopsis: Stream connected to standard input of the process.

Declaration: `Property Input : TOutputPipeStream`

Visibility: public

Access: Read

Description: `Input` is a stream which is connected to the process' standard input file handle. Anything written to this stream can be read by the process.

The `Input` stream is only instantiated when the `poUsePipes` flag is used in Options (1440).

Note that writing to the stream may cause the calling process to be suspended when the created process is not reading from it's input, or to cause errors when the process has terminated.

See also: TProcess.OutPut (1433), TProcess.StdErr (1434), TProcess.Options (1440), TProcessOption (1421)

56.6.23 TPROCESS.Output

Synopsis: Stream connected to standard output of the process.

Declaration: `Property Output : TInputPipeStream`

Visibility: public

Access: Read

Description: `Output` is a stream which is connected to the process' standard output file handle. Anything written to standard output by the created process can be read from this stream.

The `Output` stream is only instantiated when the `poUsePipes` flag is used in Options (1440).

The `Output` stream also contains any data written to standard diagnostic output (`stderr`) when the `poStdErrToOutPut` flag is used in Options (1440).

Note that reading from the stream may cause the calling process to be suspended when the created process is not writing anything to standard output, or to cause errors when the process has terminated.

See also: TProcess.InPut (1433), TProcess.StdErr (1434), TProcess.Options (1440), TProcessOption (1421)

56.6.24 TPROCESS.Stderr

Synopsis: Stream connected to standard diagnostic output of the process.

Declaration: `Property Stderr : TInputPipeStream`

Visibility: public

Access: Read

Description: `StdErr` is a stream which is connected to the process' standard diagnostic output file handle (`StdErr`). Anything written to standard diagnostic output by the created process can be read from this stream.

The `StdErr` stream is only instantiated when the `poUsePipes` flag is used in Options (1440).

The `Output` stream equals the Output (1433) when the `poStdErrToOutPut` flag is used in Options (1440).

Note that reading from the stream may cause the calling process to be suspended when the created process is not writing anything to standard output, or to cause errors when the process has terminated.

See also: TProcess.InPut (1433), TProcess.Output (1433), TProcess.Options (1440), TProcessOption (1421)

56.6.25 TPROCESS.ExitStatus

Synopsis: Exit status of the process.

Declaration: `Property ExitStatus : Integer`

Visibility: public

Access: Read

Description: `ExitStatus` contains the exit status as reported by the OS for the process when it stopped executing: Normally, this is the exit code of the process.

The value of this property is only meaningful when the process has finished executing. If it is not yet running then the value is -1. (it was zero in earlier versions of FPC)

See also: TProcess.Running (1441), TProcess.Terminate (1430), TProcess.ExitCode (1435)

56.6.26 TPROCESS.ExitCode

Synopsis: Exit code of the process.

Declaration: `Property ExitCode : Integer`

Visibility: public

Access: Read

Description: `ExitCode` is the actual exit code of the process. On UNIX, this may differ from the ExitStatus (1420) value if the process was terminated by a signal: in that case `ExitStatus` is the raw exit status as reported by one of the UNIX `Wait` command, and `ExitCode` is the exit code reported by the program.

See also: TProcess.ExitStatus (1434), TProcess.Running (1441), TProcess.WaitOnExit (1430), TProcess.Terminate (1430)

56.6.27 TPROCESS.InheritHandles

Synopsis: Should the created process inherit the open handles of the current process.

Declaration: `Property InheritHandles : Boolean`

Visibility: public

Access: Read,Write

Description: `InheritHandles` determines whether the created process inherits the open handles of the current process (value `True`) or not (`False`).

On Unix, setting this variable has no effect.

See also: TProcess.InPut (1433), TProcess.Output (1433), TProcess.StdErr (1434)

56.6.28 TPROCESS.OnRunCommandEvent

Synopsis: Event handler, called when `RunCommandLoop` is executing.

Declaration: `Property OnRunCommandEvent : TOnRunCommandEvent`

Visibility: public

Access: Read,Write

Description: `OnRunCommandEvent` is a progress report callback, called at various stages of the TProcess.RunCommandLoop (1431) call and when an exception occurs.

See also: TProcess.RunCommandLoop (1431)

56.6.29 TPROCESS.RunCommandSleepTime

Synopsis: Sleep time between attempts to collect data.

Declaration: `Property RunCommandSleepTime : Integer`

Visibility: public

Access: Read,Write

Description: Sleep time between attempts to collect data.

56.6.30 TPROCESS.OnForkEvent

Synopsis: Event triggered after fork occurred on Linux.

Declaration: `Property OnForkEvent : TProcessForkEvent`

Visibility: public

Access: Read,Write

Description: `OnForkEvent` is triggered after the fpFork (**??**)call in the child process. It can be used to e.g. close file descriptors and make changes to other resources before the fpexecv (**??**) call. This event is not used on windows.

See also: Output (1433), Input (1433), StdErr (1434), CloseInput (1429), CloseStdErr (1429), TProcessForkEvent (1421)

56.6.31 TPROCESS.PipeBufferSize

Synopsis: Buffer size to be used when using pipes.

Declaration: `Property PipeBufferSize : Cardinal`

Visibility: published

Access: Read,Write

Description: `PipeBufferSize` indicates the buffer size used when creating pipes (when `soUsePipes` is specified in `Options`). This option is not respected on all platforms (currently only Windows uses this).

See also: #fcl.pipes.CreatePipeHandles (1408)

56.6.32 TPROCESS.Active

Synopsis: Start or stop the process.

Declaration: `Property Active : Boolean`

Visibility: published

Access: Read,Write

Description: `Active` starts the process if it is set to `True`, or terminates the process if set to `False`. It's mostly intended for use in an IDE.

See also: TProcess.Execute (1428), TProcess.Terminate (1430)

56.6.33 TPROCESS.ApplicationName

Synopsis: Name of the application to start (deprecated).

Declaration: `Property ApplicationName : TProcessString; deprecated;`

Visibility: published

Access: Read,Write

Description: `ApplicationName` is an alias for TProcess.CommandLine (1437). It's mostly for use in the Windows `CreateProcess` call. If `CommandLine` is not set, then `ApplicationName` will be used instead.

`ApplicationName` is deprecated. New code should use Executable (1437) instead, and leave `ApplicationName` empty.

See also: TProcess.CommandLine (1437), TProcess.Executable (1437), TProcess.Parameters (1438)

56.6.34 TPROCESS.CommandLine

Synopsis: Command-line to execute (deprecated).

Declaration: `Property CommandLine : TProcessString; deprecated;`

Visibility: published

Access: Read,Write

Description: `CommandLine` is deprecated. To avoid problems with command-line options with spaces in them and the quoting problems that this entails, it has been superseded by the properties TProcess.Executable (1437) and TProcess.Parameters (1438), which should be used instead of `CommandLine`. New code should leave `CommandLine` empty.

`CommandLine` is the command-line to be executed: this is the name of the program to be executed, followed by any options it should be passed.

If the command to be executed or any of the arguments contains whitespace (space, tab character, linefeed character) it should be enclosed in single or double quotes.

If no absolute pathname is given for the command to be executed, it is searched for in the `PATH` environment variable. On Windows, the current directory always will be searched first. On other platforms, this is not so.

Note that either `CommandLine` or `ApplicationName` must be set prior to calling `Execute`.

See also: TProcess.ApplicationName (1436), TProcess.Executable (1437), TProcess.Parameters (1438)

56.6.35 TPROCESS.Executable

Synopsis: Executable name. Supersedes `CommandLine` and `ApplicationName`.

Declaration: `Property Executable : TProcessString`

Visibility: published

Access: Read,Write

Description: `Executable` is the name of the executable to start. It should not contain any command-line arguments. If no path is given, it will be searched in the `PATH` environment variable.

The extension must be given, none will be added by the component itself. It may be that the OS adds the extension, but this behaviour is not guaranteed.

Arguments should be passed in TProcess.Parameters (1438).

`Executable` supersedes the TProcess.CommandLine (1437) and TProcess.ApplicationName (1436) properties, which have been deprecated. However, if either of `CommandLine` or `ApplicationName` is specified, they will be used instead of `Executable`.

See also: CommandLine (1437), ApplicationName (1436), Parameters (1438)

56.6.36 TPROCESS.Parameters

Synopsis: Command-line arguments. Supersedes `CommandLine`.

Declaration: `Property Parameters : TProcessStrings`

Visibility: published

Access: Read,Write

Description: `Parameters` contains the command-line arguments that should be passed to the program specified in Executable (1437).

Commandline arguments should be specified one per item in `Parameters`: each item in `Parameters` will be passed as a separate command-line item. It is therefor not necessary to quote whitespace in the items. As a consequence, it is not allowed to specify multiple command-line parameters in 1 item in the stringlist. If a command needs 2 options `-t` and `-s`, the following is not correct:

```
With Parameters do
  begin
  add('-t -s');
  end;
```

Instead, the code should read:

```
With Parameters do
  begin
  add('-t');
  Add('-s');
  end;
```

Remark Note that `Parameters` is ignored if either of `CommandLine` or `ApplicationName` is specified. It can only be used with `Executable`.

Remark The idea of using `Parameters` is that they are passed unmodified to the operating system. On Windows, a single command-line string must be constructed, and each parameter is surrounded by double quote characters if it contains a space. The programmer must not quote parameters with spaces.

See also: Executable (1437), CommandLine (1437), ApplicationName (1436)

56.6.37 TPROCESS.ConsoleTitle

Synopsis: Title of the console window.

Declaration: `Property ConsoleTitle : TProcessString`

Visibility: published

Access: Read,Write

Description: `ConsoleTitle` is used on Windows when executing a console application: it specifies the title caption of the console window. On other platforms, this property is currently ignored.

Changing this property after the process was started has no effect.

See also: TProcess.WindowColumns (1442), TProcess.WindowRows (1443)

56.6.38 TPROCESS.CurrentDirectory

Synopsis: Working directory of the process.

Declaration: `Property CurrentDirectory : TProcessString`

Visibility: published

Access: Read,Write

Description: `CurrentDirectory` specifies the initial working directory of the newly started process.

Changing this property after the process was started has no effect, and if the process or any of its children changes their working directory, it will not reflect this.

See also: TProcess.Environment (1439)

56.6.39 TPROCESS.Desktop

Synopsis: Desktop on which to start the process.

Declaration: `Property Desktop : string`

Visibility: published

Access: Read,Write

Description: `DeskTop` is used on Windows to determine on which desktop the process' main window should be shown. Leaving this empty means the process is started on the same desktop as the currently running process.

Changing this property after the process was started has no effect.

On UNIX, this parameter is ignored.

See also: TProcess.Input (1433), TProcess.Output (1433), TProcess.StdErr (1434)

56.6.40 TPROCESS.Environment

Synopsis: Environment variables for the new process.

Declaration: `Property Environment : TProcessStrings`

Visibility: published

Access: Read,Write

Description: `Environment` contains the complete environment for the new process; it is a list of `Name=Value` pairs, one per line. You must specify all variables, i.e. the variables defined here are *not* added to the environment of the current process.

If it is empty, the environment of the current process is passed on to the new process.

See also: TProcess.Options (1440)

56.6.41 TPROCESS.Options

Synopsis: Options to be used when starting the process.

Declaration: `Property Options : TProcessOptions`

Visibility: published

Access: Read,Write

Description: `Options` determine how the process is started. They should be set before the Execute (1428) call is made.

Table 56.7:

Option	Meaning
`poRunSuspended`	Start the process in suspended state.
`poWaitOnExit`	Wait for the process to terminate before returning.
`poUsePipes`	Use pipes to redirect standard input and output.
`poStderrToOutPut`	Redirect standard error to the standard output stream.
`poNoConsole`	Do not allow access to the console window for the process (Win32 only)
`poNewConsole`	Start a new console window for the process (Win32 only)
`poDefaultErrorMode`	Use default error handling.
`poNewProcessGroup`	Start the process in a new process group (Win32 only)
`poDebugProcess`	Allow debugging of the process (Win32 only)
`poDebugOnlyThisProcess`	Do not follow processes started by this process (Win32 only)

See also: TProcessOption (1421), TProcessOptions (1421), TProcess.Priority (1440), TProcess.StartUpOptions (1441)

56.6.42 TPROCESS.Priority

Synopsis: Priority at which the process is running.

Declaration: `Property Priority : TProcessPriority`

Visibility: published

Access: Read,Write

Description: `Priority` determines the priority at which the process is running.

Table 56.8:

Priority	Meaning
`ppHigh`	The process runs at higher than normal priority.
`ppIdle`	The process only runs when the system is idle (i.e. has nothing else to do)
`ppNormal`	The process runs at normal priority.
`ppRealTime`	The process runs at real-time priority.

Note that not all priorities can be set by any user. Usually, only users with administrative rights (the root user on Unix) can set a higher process priority.

On UNIX, the process priority is mapped on `Nice` values as follows:

Table 56.9:

Priority	Nice value
`ppHigh`	20
`ppIdle`	20
`ppNormal`	0
`ppRealTime`	-20

See also: TProcessPriority (1421)

56.6.43 TPROCESS.StartupOptions

Synopsis: Additional (Windows) startup options.

Declaration: `Property StartupOptions : TStartupOptions`

Visibility: published

Access: Read,Write

Description: `StartUpOptions` contains additional startup options, used mostly on Windows system. They determine which other window layout properties are taken into account when starting the new process.

Table 56.10:

Priority	Meaning
`suoUseShowWindow`	Use the Show Window options specified in ShowWindow (1442)
`suoUseSize`	Use the specified window sizes
`suoUsePosition`	Use the specified window sizes.
`suoUseCountChars`	Use the specified console character width.
`suoUseFillAttribute`	Use the console fill attribute specified in FillAttribute (1444).

See also: TProcess.ShowWindow (1442), TProcess.WindowHeight (1442), TProcess.WindowWidth (1444), TProcess.WindowLeft (1443), TProcess.WindowTop (1443), TProcess.WindowColumns (1442), TProcess.WindowRows (1443), TProcess.FillAttribute (1444)

56.6.44 TPROCESS.Running

Synopsis: Determines whether the process is still running.

Declaration: `Property Running : Boolean`

Visibility: published

Access: Read

Description: `Running` can be read to determine whether the process is still running.

See also: TProcess.Terminate (1430), TProcess.Active (1436), TProcess.ExitStatus (1434), TProcess.ExitCode (1435)

56.6.45 TPROCESS.ShowWindow

Synopsis: Determines how the process main window is shown (Windows only).

Declaration: `Property ShowWindow : TShowWindowOptions`

Visibility: published

Access: Read,Write

Description: `ShowWindow` determines how the process' main window is shown. It is useful only on Windows.

Table 56.11:

Option	Meaning
`swoNone`	Allow system to position the window.
`swoHIDE`	The main window is hidden.
`swoMaximize`	The main window is maximized.
`swoMinimize`	The main window is minimized.
`swoRestore`	Restore the previous position.
`swoShow`	Show the main window.
`swoShowDefault`	When showing Show the main window on a default position
`swoShowMaximized`	The main window is shown maximized
`swoShowMinimized`	The main window is shown minimized
`swoshowMinNOActive`	The main window is shown minimized but not activated
`swoShowNA`	The main window is shown but not activated
`swoShowNoActivate`	The main window is shown but not activated
`swoShowNormal`	The main window is shown normally

56.6.46 TPROCESS.WindowColumns

Synopsis: Number of columns in console window (windows only).

Declaration: `Property WindowColumns : Cardinal`

Visibility: published

Access: Read,Write

Description: `WindowColumns` is the number of columns in the console window, used to run the command in. This property is only effective if `suoUseCountChars` is specified in StartupOptions (1441)

See also: TProcess.WindowHeight (1442), TProcess.WindowWidth (1444), TProcess.WindowLeft (1443), TProcess.WindowTop (1443), TProcess.WindowRows (1443), TProcess.FillAttribute (1444), TProcess.StartupOptions (1441)

56.6.47 TPROCESS.WindowHeight

Synopsis: Height of the process main window.

Declaration: `Property WindowHeight : Cardinal`

Visibility: published

Access: Read,Write

Description: `WindowHeight` is the initial height (in pixels) of the process' main window. This property is only effective if `suoUseSize` is specified in StartupOptions (1441)

See also: TProcess.WindowWidth (1444), TProcess.WindowLeft (1443), TProcess.WindowTop (1443), TProcess.WindowColumns (1442), TProcess.WindowRows (1443), TProcess.FillAttribute (1444), TProcess.StartupOptions (1441)

56.6.48 TPROCESS.WindowLeft

Synopsis: X-coordinate of the initial window (Windows only).

Declaration: `Property WindowLeft : Cardinal`

Visibility: published

Access: Read,Write

Description: `WindowLeft` is the initial X coordinate (in pixels) of the process' main window, relative to the left border of the desktop. This property is only effective if `suoUsePosition` is specified in StartupOptions (1441)

See also: TProcess.WindowHeight (1442), TProcess.WindowWidth (1444), TProcess.WindowTop (1443), TProcess.WindowColumns (1442), TProcess.WindowRows (1443), TProcess.FillAttribute (1444), TProcess.StartupOptions (1441)

56.6.49 TPROCESS.WindowRows

Synopsis: Number of rows in console window (Windows only).

Declaration: `Property WindowRows : Cardinal`

Visibility: published

Access: Read,Write

Description: `WindowRows` is the number of rows in the console window, used to run the command in. This property is only effective if `suoUseCountChars` is specified in StartupOptions (1441)

See also: TProcess.WindowHeight (1442), TProcess.WindowWidth (1444), TProcess.WindowLeft (1443), TProcess.WindowTop (1443), TProcess.WindowColumns (1442), TProcess.FillAttribute (1444), TProcess.StartupOptions (1441)

56.6.50 TPROCESS.WindowTop

Synopsis: Y-coordinate of the initial window (Windows only).

Declaration: `Property WindowTop : Cardinal`

Visibility: published

Access: Read,Write

Description: `WindowTop` is the initial Y coordinate (in pixels) of the process' main window, relative to the top border of the desktop. This property is only effective if `suoUsePosition` is specified in StartupOptions (1441)

See also: TProcess.WindowHeight (1442), TProcess.WindowWidth (1444), TProcess.WindowLeft (1443), TProcess.WindowColumns (1442), TProcess.WindowRows (1443), TProcess.FillAttribute (1444), TProcess.StartupOptions (1441)

56.6.51 TPROCESS.WindowWidth

Synopsis: Height of the process main window (Windows only).

Declaration: `Property WindowWidth : Cardinal`

Visibility: published

Access: Read,Write

Description: `WindowWidth` is the initial width (in pixels) of the process' main window. This property is only effective if `suoUseSize` is specified in StartupOptions (1441)

See also: TProcess.WindowHeight (1442), TProcess.WindowLeft (1443), TProcess.WindowTop (1443), TProcess.WindowColumns (1442), TProcess.WindowRows (1443), TProcess.FillAttribute (1444), TProcess.StartupOptions (1441)

56.6.52 TPROCESS.FillAttribute

Synopsis: Color attributes of the characters in the console window (Windows only).

Declaration: `Property FillAttribute : Cardinal`

Visibility: published

Access: Read,Write

Description: `FillAttribute` is a WORD value which specifies the background and foreground colors of the console window.

See also: TProcess.WindowHeight (1442), TProcess.WindowWidth (1444), TProcess.WindowLeft (1443), TProcess.WindowTop (1443), TProcess.WindowColumns (1442), TProcess.WindowRows (1443), TProcess.StartupOptions (1441)

56.6.53 TPROCESS.XTermProgram

Synopsis: XTerm program to use (UNIX only).

Declaration: `Property XTermProgram : string`

Visibility: published

Access: Read,Write

Description: `XTermProgram` can be used to specify the console program to use when `poConsole` is specified in TProcess.Options (1440).

If none is specified, DetectXTerm (1424) is used to detect the terminal program to use. the list specified in `TryTerminals` is tried. If none is found, then the `DESKTOP_SESSION` environment variable is examined:

kde konsole is used if it is found.

gnome gnome-terminal is used if it is found

windowmaker aterm or xterm are used if found.

If after all this, no terminal is found, then a list of default programs is tested: 'x-terminal-emulator','xterm','aterm','wterm','rxvt

See also: TProcess.Options (1440), DetectXTerm (1424)

Chapter 57

Reference for unit 'RttiUtils'

57.1 Used units

Table 57.1: Used units by unit 'RttiUtils'

Name	Page
Classes	??
StrUtils	??
System	??
sysutils	??
TypInfo	??

57.2 Overview

The rttiutils unit is a unit providing simplified access to the RTTI information from published properties using the TPropInfoList (1448) class. This access can be used when saving or restoring form properties at runtime, or for persisting other objects whose RTTI is available: the TPropsStorage (1450) class can be used for this. The implementation is based on the apputils unit from `RXLib` by *AO ROSNO* and *Master-Bank*

57.3 Constants, types and variables

57.3.1 Constants

```
sPropNameDelimiter : string = '_'
```

Separator used when constructing section/key names.

57.3.2 Types

```
TEraseSectEvent = procedure(const ASection: string) of object
```

TEraseSectEvent is used by TPropsStorage (1450) to clear a storage section, in a .ini file like fashion: The call should remove all keys in the section ASection, and remove the section from storage.

```
TFindComponentEvent = function(const Name: string) : TComponent
```

TFindComponentEvent should return the component instance for the component with name path Name. The name path should be relative to the global list of loaded components.

```
TPropStorageOption = (psoAlwaysStoreStringsCount)
```

Table 57.2: Enumeration values for type TPropStorageOption

Value	Explanation
psoAlwaysStoreStringsCount	Always store the count of strings. Default is not to store the count.

TPropStorageOption is the enumeration type used in the TPropsStorage.Options (1453) property of TPropsStorage (1450)

```
TPropStorageOptions = Set of TPropStorageOption
```

TPropStorageOptions is the set of TPropStorageOption used in TPropsStorage.Options (1453).

```
TReadStrEvent = function(const ASection: string; const Item: string
  ;
                         const Default: string) : string of
  object
```

TReadStrEvent is used by TPropsStorage (1450) to read strings from a storage mechanism, in a .ini file like fashion: The call should read the string in ASection with key Item, and if it does not exist, Default should be returned.

```
TWriteStrEvent = procedure(const ASection: string; const Item: string
  ;
                           const Value: string) of object
```

TWriteStrEvent is used by TPropsStorage (1450) to write strings to a storage mechanism, in a .ini file like fashion: The call should write the string Value in ASection with key Item. The section and key should be created if they didn't exist yet.

57.3.3 Variables

```
FindGlobalComponentCallBack : TFindComponentEvent
```

FindGlobalComponentCallBack is called by UpdateStoredList (1447) whenever it needs to resolve component references. It should be set to a routine that locates a loaded component in the global list of loaded components.

57.4 Procedures and functions

57.4.1 CreateStoredItem

Synopsis: Concatenates component and property name.

Declaration:
```
function CreateStoredItem(const CompName: string;
                          const PropName: string) : string
```

Visibility: default

Description: `CreateStoredItem` concatenates `CompName` and `PropName` if they are both empty. The names are separated by a dot (.) character. If either of the names is empty, an empty string is returned.

This function can be used to create items for the list of properties such as used in UpdateStoredList (1447), TPropsStorage.StoreObjectsProps (1452) or TPropsStorage.LoadObjectsProps (1452).

See also: ParseStoredItem (1447), UpdateStoredList (1447), TPropsStorage.StoreObjectsProps (1452), TPropsStorage.LoadObjectsProps (1452)

57.4.2 ParseStoredItem

Synopsis: Split a property reference to component reference and property name.

Declaration:
```
function ParseStoredItem(const Item: string; var CompName: string;
                         var PropName: string) : Boolean
```

Visibility: default

Description: `ParseStoredItem` parses the property reference `Item` and splits it in a reference to a component (returned in `CompName`) and a name of a property (returned in `PropName`). This function basically does the opposite of CreateStoredItem (1447). Note that both names should be non-empty, i.e., at least 1 dot character must appear in `Item`.

Errors: If an error occurred during parsing, `False` is returned.

See also: CreateStoredItem (1447), UpdateStoredList (1447), TPropsStorage.StoreObjectsProps (1452), TPropsStorage.LoadObjectsProps (1452)

57.4.3 UpdateStoredList

Synopsis: Update a stringlist with object references.

Declaration:
```
procedure UpdateStoredList(AComponent: TComponent;
                           AStoredList: TStrings; FromForm: Boolean)
```

Visibility: default

Description: `UpdateStoredList` will parse the strings in `AStoredList` using ParseStoredItem (1447) and will replace the `Objects` properties with the instance of the object whose name each property path in the list refers to. If `FromForm` is `True`, then all instances are searched relative to `AComponent`, i.e. they must be owned by `AComponent`. If `FromForm` is `False` the instances are searched in the global list of streamed components. (the FindGlobalComponentCallBack (1446) callback must be set for the search to work correctly in this case)

If a component cannot be found, the reference string to the property is removed from the stringlist.

Errors: If `AComponent` is `Nil`, an exception may be raised.

See also: ParseStoredItem (1447), TPropsStorage.StoreObjectsProps (1452), TPropsStorage.LoadObjectsProps (1452), FindGlobalComponentCallBack (1446)

57.5 TPropInfoList

57.5.1 Description

`TPropInfoList` is a class which can be used to maintain a list with information about published properties of a class (or an instance). It is used internally by TPropsStorage (1450)

See also: TPropsStorage (1450)

57.5.2 Method overview

Page	Method	Description
1449	Contains	Check whether a certain property is included.
1448	Create	Create a new instance of `TPropInfoList`.
1449	Delete	Delete property information from the list.
1448	Destroy	Remove the `TPropInfoList` instance from memory.
1449	Find	Retrieve property information based on name.
1449	Intersect	Intersect 2 property lists.

57.5.3 Property overview

Page	Properties	Access	Description
1450	Count	r	Number of items in the list.
1450	Items	r	Indexed access to the property type pointers.

57.5.4 TPropInfoList.Create

Synopsis: Create a new instance of `TPropInfoList`.

Declaration:
```
constructor Create(AObject: TObject; Filter: TTypeKinds;
                   Sorted: Boolean)
```

Visibility: public

Description: `Create` allocates and initializes a new instance of `TPropInfoList` on the heap. It retrieves a list of published properties from `AObject`: if `Filter` is empty, then all properties are retrieved. If it is not empty, then only properties of the kind specified in the set are retrieved. `Instance` should not be `Nil`

See also: Destroy (1448)

57.5.5 TPropInfoList.Destroy

Synopsis: Remove the `TPropInfoList` instance from memory.

Declaration:
```
destructor Destroy;  Override
```

Visibility: public

Description: `Destroy` cleans up the internal structures maintained by `TPropInfoList` and then calls the inherited `Destroy`.

See also: Create (1448)

57.5.6 TPropInfoList.Contains

Synopsis: Check whether a certain property is included.

Declaration: `function Contains(P: PPropInfo) : Boolean`

Visibility: public

Description: `Contains` checks whether `P` is included in the list of properties, and returns `True` if it does. If `P` cannot be found, `False` is returned.

See also: Find (1449), Intersect (1449)

57.5.7 TPropInfoList.Find

Synopsis: Retrieve property information based on name.

Declaration: `function Find(const AName: string) : PPropInfo`

Visibility: public

Description: `Find` returns a pointer to the type information of the property `AName`. If no such information is available, the function returns `Nil`. The search is performed case insensitive.

See also: Intersect (1449), Contains (1449)

57.5.8 TPropInfoList.Delete

Synopsis: Delete property information from the list.

Declaration: `procedure Delete(Index: Integer)`

Visibility: public

Description: `Delete` deletes the property information at position `Index` from the list. It's mainly of use in the Intersect (1449) call.

Errors: No checking on the validity of `Index` is performed.

See also: Intersect (1449)

57.5.9 TPropInfoList.Intersect

Synopsis: Intersect 2 property lists.

Declaration: `procedure Intersect(List: TPropInfoList)`

Visibility: public

Description: `Intersect` reduces the list of properties to the ones also contained in `List`, i.e. all properties which are not also present in `List` are removed.

See also: Delete (1449), Contains (1449)

57.5.10 TPropInfoList.Count

Synopsis: Number of items in the list.

Declaration: `Property Count : Integer`

Visibility: public

Access: Read

Description: `Count` is the number of property type pointers in the list.

See also: Items (1450)

57.5.11 TPropInfoList.Items

Synopsis: Indexed access to the property type pointers.

Declaration: `Property Items[Index: Integer]: PPropInfo; default`

Visibility: public

Access: Read

Description: `Items` provides access to the property type pointers stored in the list. `Index` runs from 0 to `Count-1`.

See also: Count (1450)

57.6 TPropsStorage

57.6.1 Description

`TPropsStorage` provides a mechanism to store properties from any class which has published properties (usually a `TPersistent` descendent) in a storage mechanism.

`TPropsStorage` does not handle the storage by itself, instead, the storage is handled through a series of callbacks to read and/or write strings. Conversion of property types to string is handled by `TPropsStorage` itself: all that needs to be done is set the 3 handlers. The storage mechanism is assumed to have the structure of an .ini file : sections with key/value pairs. The three callbacks should take this into account, but they do not need to create an actual .ini file.

See also: TPropInfoList (1448)

57.6.2 Method overview

Page	Method	Description
1451	LoadAnyProperty	Load a property value.
1452	LoadObjectsProps	Load a list of component properties.
1452	LoadProperties	Load a list of properties.
1451	StoreAnyProperty	Store a property value.
1452	StoreObjectsProps	Store a list of component properties.
1451	StoreProperties	Store a list of properties.

57.6.3 Property overview

Page	Properties	Access	Description
1453	AObject	rw	Object to load or store properties from.
1455	OnEraseSection	rw	Erase a section in storage.
1454	OnReadString	rw	Read a string value from storage.
1454	OnWriteString	rw	Write a string value to storage.
1453	Options	rw	Options to take into account when saving or loading properties from the storage.
1454	Prefix	rw	Prefix to use in storage.
1454	Section	rw	Section name for storage.

57.6.4 TPropsStorage.StoreAnyProperty

Synopsis: Store a property value.

Declaration: `procedure StoreAnyProperty(PropInfo: PPropInfo)`

Visibility: public

Description: `StoreAnyProperty` stores the property with information specified in `PropInfo` in the storage mechanism. The property value is retrieved from the object instance specified in the AObject (1453) property of `TPropsStorage`.

Errors: If the property pointer is invalid or `AObject` is invalid, an exception will be raised.

See also: AObject (1453), LoadAnyProperty (1451), LoadProperties (1452), StoreProperties (1451)

57.6.5 TPropsStorage.LoadAnyProperty

Synopsis: Load a property value.

Declaration: `procedure LoadAnyProperty(PropInfo: PPropInfo)`

Visibility: public

Description: `LoadAnyProperty` loads the property with information specified in `PropInfo` from the storage mechanism. The value is then applied to the object instance specified in the AObject (1453) property of `TPropsStorage`.

Errors: If the property pointer is invalid or `AObject` is invalid, an exception will be raised.

See also: AObject (1453), StoreAnyProperty (1451), LoadProperties (1452), StoreProperties (1451)

57.6.6 TPropsStorage.StoreProperties

Synopsis: Store a list of properties.

Declaration: `procedure StoreProperties(PropList: TStrings)`

Visibility: public

Description: `StoreProperties` stores the values of all properties in `PropList` in the storage mechanism. The list should contain names of published properties of the AObject (1453) object.

Errors: If an invalid property name is specified, an exception will be raised.

See also: AObject (1453), StoreAnyProperty (1451), LoadProperties (1452), LoadAnyProperty (1451)

57.6.7 TPropsStorage.LoadProperties

Synopsis: Load a list of properties.

Declaration: `procedure LoadProperties(PropList: TStrings)`

Visibility: public

Description: `LoadProperties` loads the values of all properties in `PropList` from the storage mechanism. The list should contain names of published properties of the AObject (1453) object.

Errors: If an invalid property name is specified, an exception will be raised.

See also: AObject (1453), StoreAnyProperty (1451), StoreProperties (1451), LoadAnyProperty (1451)

57.6.8 TPropsStorage.LoadObjectsProps

Synopsis: Load a list of component properties.

Declaration: `procedure LoadObjectsProps(AComponent: TComponent; StoredList: TStrings)`

Visibility: public

Description: `LoadObjectsProps` loads a list of component properties, relative to `AComponent`: the names of the component properties to load are specified as follows:

```
ComponentName1.PropertyName
ComponentName2.Subcomponent1.PropertyName
```

The component instances will be located relative to `AComponent`, and must therefore be names of components owned by `AComponent`, followed by a valid property of these components. If the componentname is missing, the property name will be assumed to be a property of `AComponent` itself.

The `Objects` property of the stringlist should be filled with the instances of the components the property references refer to: they can be filled with the UpdateStoredList (1447) call.

For example, to load the checked state of a checkbox named 'CBCheckMe' and the caption of a button named 'BPressMe', both owned by a form, the following strings should be passed:

```
CBCheckMe.Checked
BPressMe.Caption
```

and the `ACompontent` should be the form component that owns the button and checkbox.

Note that this call removes the value of the AObject (1453) property.

Errors: If an invalid component is specified, an exception will be raised.

See also: UpdateStoredList (1447), StoreObjectsProps (1452), LoadProperties (1452), LoadAnyProperty (1451)

57.6.9 TPropsStorage.StoreObjectsProps

Synopsis: Store a list of component properties.

Declaration:
```
procedure StoreObjectsProps(AComponent: TComponent;
                            StoredList: TStrings)
```

Visibility: public

Description: `StoreObjectsProps` stores a list of component properties, relative to `AComponent`: the names of the component properties to store are specified as follows:

```
ComponentName1.PropertyName
ComponentName2.Subcomponent1.PropertyName
```

The component instances will be located relative to `AComponent`, and must therefore be names of components owned by `AComponent`, followed by a valid property of these components. If the componentname is missing, the property name will be assumed to be a property of `AComponent` itself.

The `Objects` property of the stringlist should be filled with the instances of the components the property references refer to: they can be filled with the UpdateStoredList (1447) call.

For example, to store the checked state of a checkbox named 'CBCheckMe' and the caption of a button named 'BPressMe', both owned by a form, the following strings should be passed:

```
CBCheckMe.Checked
BPressMe.Caption
```

and the `ACompontent` should be the form component that owns the button and checkbox.

Note that this call removes the value of the AObject (1453) property.

See also: UpdateStoredList (1447), LoadObjectsProps (1452), LoadProperties (1452), LoadAnyProperty (1451)

57.6.10 TPropsStorage.Options

Synopsis: Options to take into account when saving or loading properties from the storage.

Declaration: `Property Options : TPropStorageOptions`

Visibility: public

Access: Read,Write

Description: `Options` can be used to tweak the behaviour of `TPropsStorage` when it loads or saves data to the storage. Currently the following options are available

psoAlwaysStoreStringsCountAlways store the count of strings. Default is not to store the count.

See also: TPropStorageOptions (1446), TPropStorageOption (1446)

57.6.11 TPropsStorage.AObject

Synopsis: Object to load or store properties from.

Declaration: `Property AObject : TObject`

Visibility: public

Access: Read,Write

Description: `AObject` is the object instance whose properties will be loaded or stored with any of the methods in the `TPropsStorage` class. Note that a call to StoreObjectProps (1452) or LoadObjectProps (1452) will destroy any value that this property might have.

See also: LoadProperties (1452), LoadAnyProperty (1451), StoreProperties (1451), StoreAnyProperty (1451), StoreObjectProps (1452), LoadObjectProps (1452)

57.6.12 TPropsStorage.Prefix

Synopsis: Prefix to use in storage.

Declaration: `Property Prefix : string`

Visibility: public

Access: Read,Write

Description: `Prefix` is prepended to all property names to form the key name when writing a property to storage, or when reading a value from storage. This is useful when storing properties of multiple forms in a single section.

See also: TPropsStorage.Section (1454)

57.6.13 TPropsStorage.Section

Synopsis: Section name for storage.

Declaration: `Property Section : string`

Visibility: public

Access: Read,Write

Description: `Section` is used as the section name when writing values to storage. Note that when writing properties of subcomponents, their names will be appended to the value specified here.

See also: TPropsStorage.Section (1454)

57.6.14 TPropsStorage.OnReadString

Synopsis: Read a string value from storage.

Declaration: `Property OnReadString : TReadStrEvent`

Visibility: public

Access: Read,Write

Description: `OnReadString` is the event handler called whenever `TPropsStorage` needs to read a string from storage. It should be set whenever properties need to be loaded, or an exception will be raised.

See also: OnWriteString (1454), OnEraseSection (1455), TReadStrEvent (1446)

57.6.15 TPropsStorage.OnWriteString

Synopsis: Write a string value to storage.

Declaration: `Property OnWriteString : TWriteStrEvent`

Visibility: public

Access: Read,Write

Description: `OnWriteString` is the event handler called whenever `TPropsStorage` needs to write a string to storage. It should be set whenever properties need to be stored, or an exception will be raised.

See also: OnReadString (1454), OnEraseSection (1455), TWriteStrEvent (1446)

57.6.16 TPropsStorage.OnEraseSection

Synopsis: Erase a section in storage.

Declaration: `Property OnEraseSection : TEraseSectEvent`

Visibility: public

Access: Read,Write

Description: `OnEraseSection` is the event handler called whenever `TPropsStorage` needs to clear a complete storage section. It should be set whenever stringlist properties need to be stored, or an exception will be raised.

See also: OnReadString (1454), OnWriteString (1454), TEraseSectEvent (1445)

Chapter 58

Reference for unit 'simpleipc'

58.1 Used units

Table 58.1: Used units by unit 'simpleipc'

Name	Page
Classes	??
Contnrs	284
syncobjs	1601
System	??
sysutils	??

58.2 Overview

The SimpleIPC unit provides classes to implement a simple, one-way IPC mechanism using string messages. It provides a TSimpleIPCServer (1473) component for the server, and a TSimpleIPCClient (1470) component for the client. The components are cross-platform, and should work both on Windows and UNIX-like systems.

The Unix implementation of the SimpleIPC unit uses file-based sockets. It will attempt to clean up any registered server socket files that were not removed cleanly.

It does this in the unit finalization code. It does not install a signal handler by itself, that is the task of the programmer. But program crashes (access violations and such) that are handled by the RTL will be handled gracefully.

This also means that if the process is killed with the KILL signal, it has no chance of removing the files (KILL signals cannot be caught), in which case socket files may remain in the file system. However, the client code attempts to cater for this and will remove the stale sockets if it detects them.

Under Windows, the communication is done through `WM_COPYDATA` messages. Starting from Windows Vista it is forbidden to send messages between service applications and desktop applications, so a SimpleIPC client in a desktop application cannot connect to a SimpleIPC server in a service application and vice versa.

58.3 Constants, types and variables

58.3.1 Resource strings

```
SErrActive =
  'This operation is illegal when the server is active.'
```

Error message if client/server is active.

```
SErrInActive =
  'This operation is illegal when the server is inactive.'
```

Error message if client/server is not active.

```
SErrMessageQueueOverflow = 'Message queue overflow (limit %s)'
```

Too many messages in the message queue.

```
SErrServerNotActive = 'Server with ID %s is not active.'
```

Error message if server is not active.

```
SErrThreadContext =
  'This operation is illegal outside of IPC thread context.'
```

Thread context error message.

```
SErrThreadFailure = 'IPC thread failure.'
```

Thread failure message.

58.3.2 Constants

```
MsgVersion = 1
```

Current version of the messaging protocol.

```
mtString = 1
```

String message type.

```
mtUnknown = 0
```

Unknown message type.

58.3.3 Types

```
TIPCClientCommClass = Class of TIPCClientComm
```

`TIPCClientCommClass` is used by TSimpleIPCClient (1470) to decide which kind of communication channel to set up.

```
TIPCMessageOverflowAction = (ipcmoaNone,ipcmoaDiscardOld,
  ipcmoaDiscardNew,ipcmoaError)
```

Table 58.2: Enumeration values for type TIPCMessageOverflowAction

Value	Explanation
ipcmoaDiscardNew	Discard the new message.
ipcmoaDiscardOld	Discard the oldest message.
ipcmoaError	Raise an error.
ipcmoaNone	Do nothing, just add the message.

`TIPCMessageOverflowAction` describes what will happen if the message queue hits the size limit for the queue.

Do nothing, just add the message.

Discard the oldest message.

Discard the new message.

```
TIPCServerCommClass = Class of TIPCServerComm
```

`TIPCServerCommClass` is used by TSimpleIPCServer (1473) to decide which kind of communication channel to set up.

```
TMessageQueueEvent = procedure(Sender: TObject; Msg: TIPCServerMsg
  )
                                  of object
```

`TMessageQueueEvent` is the signature of the event handler that is executed when a new message arrives on the server and the queue is full, and maxaction is `ipcmoaError`.

```
TMessageType = LongInt
```

`TMessageType` is provided for backward compatibility with earlier versions of the simpleipc unit.

58.3.4 Variables

```
DefaultIPCClientClass : TIPCClientCommClass = Nil
```

`DefaultIPCClientClass` is filled with a class pointer indicating which kind of communication protocol class should be instantiated by the TSimpleIPCClient (1470) class. It is set to a default value by the default implementation in the SimpleIPC unit, but can be set to another class if another method of transport is desired. (it should match the communication protocol used by the server, obviously).

```
DefaultIPCMessageOverflowAction : TIPCMessageOverflowAction = TSimpleIPCServer
  .DefaultMaxAction
```

`DefaultIPCMessageOverflowAction` is the default for the message queue overflow action when a new queue is made.

```
DefaultIPCMessageQueueLimit : Integer = TSimpleIPCServer
  .DefaultMaxQueue
```

DefaultIPCMessageOverflowAction is the default for the maximum message queue size when a new queue is made. A zero size means no limit.

```
DefaultIPCServerClass : TIPCServerCommClass = Nil
```

DefaultIPCServerClass is filled with a class pointer indicating which kind of communication protocol class should be instantiated by the TSimpleIPCServer (1473) class. It is set to a default value by the default implementation in the SimpleIPC unit, but can be set to another class if another method of transport is desired.

58.4 TMsgHeader

```
TMsgHeader = packed record
  Version : Byte;
  MsgType : TMessageType
  ;
  MsgLen : Integer;
end
```

TMsgHeader is used internally by the IPC client and server components to transmit data. The Version field denotes the protocol version. The MsgType field denotes the type of data (mtString for string messages), and MsgLen is the length of the message which will follow.

58.5 EIPCError

58.5.1 Description

EIPCError is the exception used by the various classes in the SimpleIPC unit to report errors.

58.6 TIPCClientComm

58.6.1 Description

TIPCClientComm is an abstract component which implements the client-side communication protocol. The behaviour expected of this class must be implemented in a platform-dependent descendent class.

The TSimpleIPCClient (1470) class does not implement the messaging protocol by itself. Instead, it creates an instance of a (platform dependent) descendent of TIPCClientComm which handles the internals of the communication protocol.

The server side of the messaging protocol is handled by the TIPCServerComm (1462) component. The descendent components must always be implemented in pairs.

See also: TSimpleIPCClient (1470), TIPCServerComm (1462), TSimpleIPCServer (1473)

58.6.2 Method overview

Page	Method	Description
1460	Connect	Connect to the server.
1460	Create	Create a new instance of the `TIPCClientComm`.
1460	Disconnect	Disconnect from the server.
1461	SendMessage	Send a message.
1461	ServerRunning	Check if the server is running.

58.6.3 Property overview

Page	Properties	Access	Description
1461	Owner	r	`TSimpleIPCClient` instance for which communication must be handled.

58.6.4 TIPCClientComm.Create

Synopsis: Create a new instance of the `TIPCClientComm`.

Declaration:
```
constructor Create(AOwner: TSimpleIPCClient);  Virtual
```

Visibility: public

Description: `Create` instantiates a new instance of the `TIPCClientComm` class, and stores the `AOwner` reference to the TSimpleIPCClient (1470) instance for which it will handle communication. It can be retrieved later using the Owner (1461) property.

See also: Owner (1461), TSimpleIPCClient (1470)

58.6.5 TIPCClientComm.Connect

Synopsis: Connect to the server.

Declaration:
```
procedure Connect;  Virtual;  Abstract
```

Visibility: public

Description: `Connect` must establish a communication channel with the server. The server endpoint must be constructed from the ServerID (1469) and ServerInstance (1472) properties of the owning TSimpleIPCClient (1470) instance.

`Connect` is called by the TSimpleIPCClient.Connect (1471) call or when the Active (1469) property is set to `True`

Messages can be sent only after `Connect` was called successfully.

Errors: If the connection setup fails, or the connection was already set up, then an exception may be raised.

See also: TSimpleIPCClient.Connect (1471), Active (1469), Disconnect (1460)

58.6.6 TIPCClientComm.Disconnect

Synopsis: Disconnect from the server.

Declaration:
```
procedure Disconnect;  Virtual;  Abstract
```

Visibility: public

Description: `Disconnect` closes the communication channel with the server. Any calls to `SendMessage` are invalid after `Disconnect` was called.

`Disconnect` is called by the TSimpleIPCClient.Disconnect (1471) call or when the Active (1469) property is set to `False`.

Messages can no longer be sent after `Disconnect` was called.

Errors: If the connection shutdown fails, or the connection was already shut down, then an exception may be raised.

See also: TSimpleIPCClient.Disconnect (1471), Active (1469), Connect (1460)

58.6.7 TIPCClientComm.ServerRunning

Synopsis: Check if the server is running.

Declaration:
```
function ServerRunning : Boolean;  Virtual;  Abstract
```

Visibility: public

Description: `ServerRunning` returns `True` if the server endpoint for the communication channel can be found, or `False` if not. The server endpoint is obtained from the `ServerID` property in the owning TSimpleIPCClient (1470) component.

See also: ServerID (1469), InstanceID (1477)

58.6.8 TIPCClientComm.SendMessage

Synopsis: Send a message.

Declaration:
```
procedure SendMessage(MsgType: TMessageType; Stream: TStream);  Virtual
                      ;  Abstract
```

Visibility: public

Description: `SendMessage` should deliver the message with type `MsgType` and data in `Stream` to the server. It should not return until the message was delivered.

Errors: If the delivery of the message fails, an exception will be raised.

58.6.9 TIPCClientComm.Owner

Synopsis: `TSimpleIPCClient` instance for which communication must be handled.

Declaration:
```
Property Owner : TSimpleIPCClient
```

Visibility: public

Access: Read

Description: `Owner` is the TSimpleIPCClient (1470) instance for which the communication must be handled. It cannot be changed, and must be specified when the `TIPCClientComm` instance is created.

See also: TSimpleIPCClient (1470), TIPCClientComm.Create (1460)

58.7 TIPCServerComm

58.7.1 Description

`TIPCServerComm` is an abstract component which implements the server-side communication protocol. The behaviour expected of this class must be implemented in a platform-dependent descendent class.

The TSimpleIPCServer (1473) class does not implement the messaging protocol by itself. Instead, it creates an instance of a (platform dependent) descendent of `TIPCServerComm` which handles the internals of the communication protocol.

The client side of the messaging protocol is handled by the TIPCClientComm (1459) component. The descendent components must always be implemented in pairs.

See also: TSimpleIPCServer (1473), TIPCClientComm (1459)

58.7.2 Method overview

Page	Method	Description
1462	Create	Create a new instance of the communication handler.
1463	PeekMessage	See if a message is available.
1463	ReadMessage	Read message from the channel.
1462	StartServer	Start the server-side of the communication channel.
1463	StopServer	Stop the server side of the communication channel.

58.7.3 Property overview

Page	Properties	Access	Description
1464	InstanceID	r	Unique identifier for the communication channel.
1464	Owner	r	`TSimpleIPCServer` instance for which to handle transport.

58.7.4 TIPCServerComm.Create

Synopsis: Create a new instance of the communication handler.

Declaration:
```
constructor Create(AOwner: TSimpleIPCServer);  Virtual
```

Visibility: public

Description: `Create` initializes a new instance of the communication handler. It simply saves the `AOwner` parameter in the Owner (1464) property.

See also: Owner (1464)

58.7.5 TIPCServerComm.StartServer

Synopsis: Start the server-side of the communication channel.

Declaration:
```
procedure StartServer;  Virtual;  Abstract
```

Visibility: public

Description: `StartServer` sets up the server-side of the communication channel. After `StartServer` was called, a client can connect to the communication channel, and send messages to the server.

It is called when the TSimpleIPC.Active (1469) property of the TSimpleIPCServer (1473) instance is set to `True`.

If `Threaded` is `True` then a background thread is started which will check for new messages periodically (see also TSimpleIPCServer.ThreadTimeOut (1480)). The arrival of new messages can be acted upon with TSimpleIPCServer.OnMessageQueued (1478).

Errors: In case of an error, an EIPCError (1459) exception is raised.

See also: TSimpleIPCServer (1473), TSimpleIPC.Active (1469), TSimpleIPCServer.OnMessageQueued (1478), TSimpleIPCServer.ThreadTimeOut (1480)

58.7.6 TIPCServerComm.StopServer

Synopsis: Stop the server side of the communication channel.

Declaration: `procedure StopServer; Virtual; Abstract`

Visibility: public

Description: `StopServer` closes down the server-side of the communication channel. After `StartServer` was called, a client can no longer connect to the communication channel, or even send messages to the server if it was previously connected (i.e. it will be disconnected).

It is called when the TSimpleIPC.Active (1469) property of the TSimpleIPCServer (1473) instance is set to `False`.

Errors: In case of an error, an EIPCError (1459) exception is raised.

See also: TSimpleIPCServer (1473), TSimpleIPC.Active (1469)

58.7.7 TIPCServerComm.PeekMessage

Synopsis: See if a message is available.

Declaration: `function PeekMessage(Timeout: Integer) : Boolean; Virtual; Abstract`

Visibility: public

Description: `PeekMessage` can be used to see if a message is available: it returns `True` if a message is available. It will wait maximum `TimeOut` milliseconds for a message to arrive. If no message was available after this time, it will return `False`.

If a message was available, it can be read with the ReadMessage (1463) call.

See also: ReadMessage (1463)

58.7.8 TIPCServerComm.ReadMessage

Synopsis: Read message from the channel.

Declaration: `procedure ReadMessage; Virtual; Abstract`

Visibility: public

Description: `ReadMessage` reads the message for the channel, and stores the information in the data structures in the `Owner` class.

`ReadMessage` is a blocking call: if no message is available, the program will wait till a message arrives. Use PeekMessage (1463) to see if a message is available.

See also: TSimpleIPCServer (1473)

58.7.9 TIPCServerComm.Owner

Synopsis: `TSimpleIPCServer` instance for which to handle transport.

Declaration: `Property Owner : TSimpleIPCServer`

Visibility: public

Access: Read

Description: `Owner` refers to the TSimpleIPCServer (1473) instance for which this instance of `TSimpleIPCServer` handles the transport. It is specified when the `TIPCServerComm` is created.

See also: TSimpleIPCServer (1473)

58.7.10 TIPCServerComm.InstanceID

Synopsis: Unique identifier for the communication channel.

Declaration: `Property InstanceID : string`

Visibility: public

Access: Read

Description: `InstanceID` returns a textual representation which uniquely identifies the communication channel on the server. The value is system dependent, and should be usable by the client-side to establish a communication channel with this instance.

58.8 TIPCServerMsg

58.8.1 Description

`TIPCServerMsg` is an auxiliary class used in the IPC server class TSimpleIPCServer (1473). It keeps the data for 1 message. The set of messages is managed in TIPCServerMsgQueue (1466). There should normally be no need to use this class directly.

See also: TIPCServerMsgQueue (1466)

58.8.2 Method overview

Page	Method	Description
1465	Create	Create a new instance of a server message.
1465	Destroy	Destroy an instance of a server message.

58.8.3 Property overview

Page	Properties	Access	Description
1465	MsgType	rw	Message type.
1466	OwnsStream	rw	Does the message own the stream.
1465	Stream	r	Stream to store message data.
1466	StringMessage	r	String message sent by client.

58.8.4 TIPCServerMsg.Create

Synopsis: Create a new instance of a server message.

Declaration:
```
constructor Create
constructor Create(AStream: TStream; AOwnsStream: Boolean)
```

Visibility: public

Description: `Create` initializes the stream used to hold the message data.

See also: TIPCServerMsg.Destroy (1465)

58.8.5 TIPCServerMsg.Destroy

Synopsis: Destroy an instance of a server message.

Declaration:
```
destructor Destroy;  Override
```

Visibility: public

Description: `Destroy` frees the stream used to hold the message data.

See also: TIPCServerMsg.Create (1465)

58.8.6 TIPCServerMsg.Stream

Synopsis: Stream to store message data.

Declaration:
```
Property Stream : TStream
```

Visibility: public

Access: Read

Description: `Stream` contains the message data as binary data.

See also: TIPCServerMsg.MsgType (1465)

58.8.7 TIPCServerMsg.MsgType

Synopsis: Message type.

Declaration:
```
Property MsgType : TMessageType
```

Visibility: public

Access: Read,Write

Description: `MsgType` simply contains the message type. The possible message types are application defined.

See also: TIPCServerMsg.Stream (1465)

58.8.8 TIPCServerMsg.OwnsStream

Synopsis: Does the message own the stream.

Declaration: `Property OwnsStream : Boolean`

Visibility: public

Access: Read,Write

Description: `OwnsStream` can be set to true to signal that the message should release the stream when the message is destroyed. The initial value can be specified in the constructor.

See also: TIPCServerMsg.Create (1465)

58.8.9 TIPCServerMsg.StringMessage

Synopsis: String message sent by client.

Declaration: `Property StringMessage : string`

Visibility: public

Access: Read

Description: `StringMessage` is the message sent by the client as a string.

58.9 TIPCServerMsgQueue

58.9.1 Description

`TIPCServerMsgQueue` implements a message queue with FIFO characteristics. It has support for a maximum queue length (TIPCServerMsgQueue.MaxCount (1468)) and various ways of dealing with overflowing queue (TIPCServerMsgQueue.MaxAction (1468))

See also: TIPCServerMsgQueue.MaxCount (1468), TIPCServerMsgQueue.MaxAction (1468)

58.9.2 Method overview

Page	Method	Description
1467	Clear	Clear the message queue.
1467	Create	Create a new message queue instance.
1467	Destroy	Destroy server message queue instance.
1468	Pop	Remove the oldest message from the queue.
1467	Push	Add a new message to the queue.

58.9.3 Property overview

Page	Properties	Access	Description
1468	Count	r	Number of messages in the queue.
1468	MaxAction	rw	Action to take when the number of messages will exceed `MaxCount`.
1468	MaxCount	rw	Maximum number of messages in the queue, 0 for unlimited.

58.9.4 TIPCServerMsgQueue.Create

Synopsis: Create a new message queue instance.

Declaration: `constructor Create`

Visibility: public

Description: `Create` creates a list to contain the messages, and initializes TIPCServerMsgQueue.MaxCount (1468) and TIPCServerMsgQueue.MaxAction (1468) with their default values (DefaultIPCMessageQueueLimit (1459) and DefaultIPCMessageOverflowAction (1458), respectively)

Note that the messages are owned by the queue till they are popped of the queue.

See also: TIPCServerMsgQueue.MaxCount (1468), TIPCServerMsgQueue.MaxAction (1468), DefaultIPCMessageQueueLimit (1459), DefaultIPCMessageOverflowAction (1458)

58.9.5 TIPCServerMsgQueue.Destroy

Synopsis: Destroy server message queue instance.

Declaration: `destructor Destroy;  Override`

Visibility: public

Description: `Destroy` discards the remaining messages in the list and removes the message queue from memory.

See also: TIPCServerMsgQueue.Create (1467), TIPCServerMsgQueue.Clear (1467)

58.9.6 TIPCServerMsgQueue.Clear

Synopsis: Clear the message queue.

Declaration: `procedure Clear`

Visibility: public

Description: `Clear` discards the remaining messages in the list.

58.9.7 TIPCServerMsgQueue.Push

Synopsis: Add a new message to the queue.

Declaration: `procedure Push(AItem: TIPCServerMsg)`

Visibility: public

Description: `Push` verifies if the message can be added to the queue (discarding old messages depending on the setting of TIPCServerMsgQueue.MaxAction (1468)) and adds the message `AItem` to the queue.

The message `AItem` is owned by the queue until it is popped off the queue.

Errors: If the maximum queue length is reached, and the MaxAction (1468) is set to `ipcmoaError`, an exception will be raised.

See also: TIPCServerMsgQueue.MaxCount (1468), TIPCServerMsgQueue.MaxAction (1468), TIPCServerMsgQueue.Pop (1468)

58.9.8 TIPCServerMsgQueue.Pop

Synopsis: Remove the oldest message from the queue.

Declaration: `function Pop : TIPCServerMsg`

Visibility: public

Description: `Pop` removes the oldest message from the queue if there is one, and returns it. If none exists, `Nil` is returned. The caller is responsible for freeing the message instance.

Errors: None.

See also: TIPCServerMsgQueue.Push (1467)

58.9.9 TIPCServerMsgQueue.Count

Synopsis: Number of messages in the queue.

Declaration: `Property Count : Integer`

Visibility: public

Access: Read

Description: `Count` is the current number of messages in the queue.

See also: MaxCount (1468)

58.9.10 TIPCServerMsgQueue.MaxCount

Synopsis: Maximum number of messages in the queue, 0 for unlimited.

Declaration: `Property MaxCount : Integer`

Visibility: public

Access: Read,Write

Description: `MaxCount` is the maximum number of messages in the queue. When this amount is zero, the amount of messages is unlimited.

When a new message is pushed, and the Count (1468) is equal to `MaxCount`, the MaxAction (1468) property is examined to know what to do.

See also: Count (1468), MaxAction (1468)

58.9.11 TIPCServerMsgQueue.MaxAction

Synopsis: Action to take when the number of messages will exceed `MaxCount`.

Declaration: `Property MaxAction : TIPCMessageOverflowAction`

Visibility: public

Access: Read,Write

Description: `MaxAction` determines what will happen if the current Count (1468) equals MaxCount (1468) and a new message is put in the queue using Push (1467):

Do nothing, just add the message.

Discard the oldest message.

Discard the new message.

See also: TIPCServerMsgQueue.Count (1468), TIPCServerMsgQueue.MaxCount (1468), TIPCServerMsgQueue.Push (1467), TIPCMessageOverflowAction (1458)

58.10 TSimpleIPC

58.10.1 Description

`TSimpleIPC` is the common ancestor for the TSimpleIPCServer (1473) and TSimpleIPCClient (1470) classes. It implements some common properties between client and server.

See also: TSimpleIPCServer (1473), TSimpleIPCClient (1470)

58.10.2 Property overview

Page	Properties	Access	Description
1469	Active	rw	Communication channel active.
1469	ServerID	rw	Unique server identification.

58.10.3 TSimpleIPC.Active

Synopsis: Communication channel active.

Declaration: `Property Active : Boolean`

Visibility: published

Access: Read,Write

Description: `Active` can be set to `True` to set up the client or server end of the communication channel. For the server this means that the server end is set up, for the client it means that the client tries to connect to the server with ServerID (1469) identification.

See also: ServerID (1469)

58.10.4 TSimpleIPC.ServerID

Synopsis: Unique server identification.

Declaration: `Property ServerID : string`

Visibility: published

Access: Read,Write

Description: `ServerID` is the unique server identification: on the server, it determines how the server channel is set up, on the client it determines the server with which to connect.

The `ServerID` is used in filenames on some platforms, therefore you should not use characters that are not allowed or have special meaning (such as /) in filenames.

See also: Active (1469)

58.11 TSimpleIPCClient

58.11.1 Description

`TSimpleIPCClient` is the client side of the simple IPC communication protocol. The client program should create a `TSimpleIPCClient` instance, set its `ServerID` property to the unique name for the server it wants to send messages to, and then set the `Active` property to `True`.

After the connection with the server was established, messages can be sent to the server with the SendMessage (1472) or SendStringMessage (1472) calls.

See also: TSimpleIPCServer (1473), TSimpleIPC (1469), TIPCClientComm (1459)

58.11.2 Method overview

Page	Method	Description
1471	Connect	Connect to the server.
1470	Create	Create a new instance of `TSimpleIPCClient`.
1470	Destroy	Remove the `TSimpleIPCClient` instance from memory.
1471	Disconnect	Disconnect from the server.
1472	SendMessage	Send a message to the server.
1472	SendStringMessage	Send a string message to the server.
1472	SendStringMessageFmt	Send a formatted string message.
1471	ServerRunning	Check if the server is running.

58.11.3 Property overview

Page	Properties	Access	Description
1472	ServerInstance	rw	Server instance identification.

58.11.4 TSimpleIPCClient.Create

Synopsis: Create a new instance of `TSimpleIPCClient`.

Declaration: `constructor Create(AOwner: TComponent); Override`

Visibility: public

Description: `Create` instantiates a new instance of the `TSimpleIPCClient` class. It initializes the data structures needed to handle the client side of the communication.

See also: Destroy (1470)

58.11.5 TSimpleIPCClient.Destroy

Synopsis: Remove the `TSimpleIPCClient` instance from memory.

Declaration: `destructor Destroy; Override`

Visibility: public

Description: `Destroy` disconnects the client from the server if need be, and cleans up the internal data structures maintained by `TSimpleIPCClient` and then calls the inherited `Destroy`, which will remove the instance from memory.

Never call `Destroy` directly, use the `Free` method instead or the `FreeAndNil` procedure in SysUtils.

See also: Create (1470)

58.11.6 TSimpleIPCClient.Connect

Synopsis: Connect to the server.

Declaration: `procedure Connect`

Visibility: public

Description: `Connect` connects to the server indicated in the ServerID (1469) and InstanceID (1477) properties. `Connect` is called automatically if the Active (1469) property is set to `True`.

After a successful call to `Connect`, messages can be sent to the server using SendMessage (1472) or SendStringMessage (1472).

Calling `Connect` if the connection is already open has no effect.

Errors: If creating the connection fails, an EIPCError (1459) exception may be raised.

See also: ServerID (1469), InstanceID (1470), Active (1469), SendMessage (1472), SendStringMessage (1472), Disconnect (1471)

58.11.7 TSimpleIPCClient.Disconnect

Synopsis: Disconnect from the server.

Declaration: `procedure Disconnect`

Visibility: public

Description: `Disconnect` shuts down the connection with the server as previously set up with Connect (1471). `Disconnect` is called automatically if the Active (1469) property is set to `False`.

After a successful call to `Disconnect`, messages can no longer be sent to the server. Attempting to do so will result in an exception.

Calling `Disconnect` if there is no connection has no effect.

Errors: If creating the connection fails, an EIPCError (1459) exception may be raised.

See also: Active (1469), Connect (1471)

58.11.8 TSimpleIPCClient.ServerRunning

Synopsis: Check if the server is running.

Declaration: `function ServerRunning : Boolean`

Visibility: public

Description: `ServerRunning` verifies if the server indicated in the ServerID (1469) and InstanceID (1477) properties is running. It returns `True` if the server communication endpoint can be reached, `False` otherwise. This function can be called before a connection is made.

See also: Connect (1471)

58.11.9 TSimpleIPCClient.SendMessage

Synopsis: Send a message to the server.

Declaration:
```
procedure SendMessage(MsgType: TMessageType; Stream: TStream)
```

Visibility: public

Description: `SendMessage` sends a message of type `MsgType` and data from `stream` to the server. The client must be connected for this call to work.

Errors: In case an error occurs, or there is no connection to the server, an EIPCError (1459) exception is raised.

See also: Connect (1471), SendStringMessage (1472)

58.11.10 TSimpleIPCClient.SendStringMessage

Synopsis: Send a string message to the server.

Declaration:
```
procedure SendStringMessage(const Msg: string)
procedure SendStringMessage(MsgType: TMessageType; const Msg: string)
```

Visibility: public

Description: `SendStringMessage` sends a string message with type `MsgTyp` and data `Msg` to the server. This is a convenience function: a small wrapper around the SendMessage (1472) method

Errors: Same as for `SendMessage`.

See also: SendMessage (1472), Connect (1471), SendStringMessageFmt (1472)

58.11.11 TSimpleIPCClient.SendStringMessageFmt

Synopsis: Send a formatted string message.

Declaration:
```
procedure SendStringMessageFmt(const Msg: string; Args: Array of const)
procedure SendStringMessageFmt(MsgType: TMessageType;
                               const Msg: string; Args: Array of const)
```

Visibility: public

Description: `SendStringMessageFmt` sends a string message with type `MsgTyp` and message formatted from `Msg` and `Args` to the server. This is a convenience function: a small wrapper around the SendStringMessage (1472) method

Errors: Same as for `SendMessage`.

See also: SendMessage (1472), Connect (1471), SendStringMessage (1472)

58.11.12 TSimpleIPCClient.ServerInstance

Synopsis: Server instance identification.

Declaration:
```
Property ServerInstance : string
```

Visibility: public

Access: Read,Write

Description: `ServerInstance` should be used in case a particular instance of the server identified with `ServerID` should be contacted. This must be used if the server has its GLobal (1477) property set to `False`, and should match the server's InstanceID (1477) property.

See also: ServerID (1469), GLobal (1477), InstanceID (1477)

58.12 TSimpleIPCServer

58.12.1 Description

`TSimpleIPCServer` is the server side of the simple IPC communication protocol. The server program should create a `TSimpleIPCServer` instance, set its ServerID (1469) property to a unique name for the system, and then set the Active (1469) property to `True` (or call StartServer (1474)).

After the server was started, it can check for availability of messages with the PeekMessage (1475) call, and read the message with ReadMessage (1475).

See also: TSimpleIPCClient (1470), TSimpleIPC (1469), TIPCServerComm (1462)

58.12.2 Method overview

Page	Method	Description
1474	Create	Create a new instance of `TSimpleIPCServer`.
1474	Destroy	Remove the `TSimpleIPCServer` instance from memory.
1475	GetMessageData	Read the data of the last message in a stream.
1475	PeekMessage	Check if a client message is available.
1475	ReadMessage	Read message from the queue.
1474	StartServer	Start the server.
1475	StopServer	Stop the server.

58.12.3 Property overview

Page	Properties	Access	Description
1477	Global	rw	Is the server reachable to all users or not.
1477	InstanceID	r	Instance ID.
1479	MaxAction	rw	Action to take when the number of messages will exceed `MaxQueue`.
1479	MaxQueue	rw	Maximum number of messages in the queue, 0 for unlimited.
1476	Message	r	Last read message.
1476	MsgData	r	Last message data.
1476	MsgType	r	Last message type.
1478	OnMessage	rw	Event triggered when a message arrives.
1478	OnMessageError	rw	Event called when a new message has arrived, and the queue is full.
1478	OnMessageQueued	rw	Event called when a new message has arrived.
1478	OnThreadError	rw	Triggered when a thread reports an error.
1476	StringMessage	r	Last message as a string.
1480	SynchronizeEvents	rw	Should events be run in the main thread ?
1479	Threaded	rw	Is the server running threaded or not ?
1477	ThreadError	r	Last thread error.
1477	ThreadExecuting	r	Is the message thread currently executing ?
1480	ThreadTimeout	rw	Timeout waiting for message.

58.12.4 TSimpleIPCServer.Create

Synopsis: Create a new instance of `TSimpleIPCServer`.

Declaration:
```
constructor Create(AOwner: TComponent);  Override
```

Visibility: public

Description: `Create` instantiates a new instance of the `TSimpleIPCServer` class. It initializes the data structures needed to handle the server side of the communication.

See also: Destroy (1474)

58.12.5 TSimpleIPCServer.Destroy

Synopsis: Remove the `TSimpleIPCServer` instance from memory.

Declaration:
```
destructor Destroy;  Override
```

Visibility: public

Description: `Destroy` stops the server, cleans up the internal data structures maintained by `TSimpleIPCServer` and then calls the inherited `Destroy`, which will remove the instance from memory.

Never call `Destroy` directly, use the `Free` method instead or the `FreeAndNil` procedure in SysUtils.

See also: Create (1474)

58.12.6 TSimpleIPCServer.StartServer

Synopsis: Start the server.

Declaration:
```
procedure StartServer
procedure StartServer(AThreaded: Boolean)
```

Visibility: public

Description: `StartServer` starts the server side of the communication channel. It is called automatically when the `Active` property is set to `True`. It creates the internal communication object (a TIPCServerComm (1462) descendent) and activates the communication channel.

The `aThreaded` property can be used to force or disable threaded mode: in threaded mode, a thread is started that automatically checks for new messages and puts them on a queue. If the argument is not specified, then the property TSimpleIPCServer.Threaded (1479) is examined to know whether to start in threaded mode or not.

After this method was called, clients can connect and send messages.

Prior to calling this method, the ServerID (1469) property must be set.

Errors: If an error occurs a EIPCError (1459) exception may be raised.

See also: TIPCServerComm (1462), Active (1469), ServerID (1469), StopServer (1475), TSimpleIPCServer.Threaded (1479)

58.12.7 TSimpleIPCServer.StopServer

Synopsis: Stop the server.

Declaration: `procedure StopServer`

Visibility: public

Description: `StopServer` stops the server side of the communication channel. It is called automatically when the `Active` property is set to `False`. It deactivates the communication channel and frees the internal communication object (a TIPCServerComm (1462) descendent).

See also: TIPCServerComm (1462), Active (1469), ServerID (1469), StartServer (1474)

58.12.8 TSimpleIPCServer.PeekMessage

Synopsis: Check if a client message is available.

Declaration: `function PeekMessage(Timeout: Integer; DoReadMessage: Boolean) : Boolean`

Visibility: public

Description: `PeekMessage` checks if a message from a client is available. It will return `True` if a message is available. The call will wait for `TimeOut` milliseconds for a message to arrive: if after `TimeOut` milliseconds, no message is available, the function will return `False`.

If `DoReadMessage` is `True` then `PeekMessage` will read the message. If it is `False`, it does not read the message. The message should then be read manually with ReadMessage (1475).

See also: ReadMessage (1475)

58.12.9 TSimpleIPCServer.ReadMessage

Synopsis: Read message from the queue.

Declaration: `function ReadMessage : Boolean`

Visibility: public

Description: `ReadMessage` will read the oldest message from the queue, and make it available in TSimpleIPCServer.MsgType (1476) and TSimpleIPCServer.MsgData (1476)

It is safe to call this even if a watch thread is started.

See also: TSimpleIPCServer.MsgType (1476), TSimpleIPCServer.MsgData (1476)

58.12.10 TSimpleIPCServer.GetMessageData

Synopsis: Read the data of the last message in a stream.

Declaration: `procedure GetMessageData(Stream: TStream)`

Visibility: public

Description: `GetMessageData` reads the data of the last message from TSimpleIPCServer.MsgData (1476) and stores it in stream `Stream`. If no data was available, the stream will be cleared.

This function will return valid data only after a successful call to ReadMessage (1475). It will also not clear the data buffer.

See also: StringMessage (1476), MsgData (1476), MsgType (1476)

58.12.11 TSimpleIPCServer.StringMessage

Synopsis: Last message as a string.

Declaration: `Property StringMessage : string`

Visibility: public

Access: Read

Description: `StringMessage` is the content of the last message as a string.

This property will contain valid data only after a successful call to ReadMessage (1475).

See also: GetMessageData (1475)

58.12.12 TSimpleIPCServer.Message

Synopsis: Last read message.

Declaration: `Property Message : TIPCServerMsg`

Visibility: public

Access: Read

Description: `Message` is the last read message (using TSimpleIPCServer.ReadMessage (1475)) from the message queue.

See also: TSimpleIPCServer.ReadMessage (1475)

58.12.13 TSimpleIPCServer.MsgType

Synopsis: Last message type.

Declaration: `Property MsgType : TMessageType`

Visibility: public

Access: Read

Description: `MsgType` contains the message type of the last message.

This property will contain valid data only after a successful call to ReadMessage (1475).

See also: ReadMessage (1475)

58.12.14 TSimpleIPCServer.MsgData

Synopsis: Last message data.

Declaration: `Property MsgData : TStream`

Visibility: public

Access: Read

Description: `MsgData` contains the actual data from the last read message. If the data is a string, then StringMessage (1476) is better suited to read the data.

This property will contain valid data only after a successful call to ReadMessage (1475).

See also: StringMessage (1476), ReadMessage (1475)

58.12.15 TSimpleIPCServer.InstanceID

Synopsis: Instance ID.

Declaration: `Property InstanceID : string`

Visibility: public

Access: Read

Description: `InstanceID` is the unique identifier for this server communication channel endpoint, and will be appended to the ServerID (1469) property to form the unique server endpoint which a client should use.

See also: ServerID (1469), Global (1477)

58.12.16 TSimpleIPCServer.ThreadExecuting

Synopsis: Is the message thread currently executing ?

Declaration: `Property ThreadExecuting : Boolean`

Visibility: public

Access: Read

Description: `ThreadExecuting` is true if the server is currently running a message loop in a thread and the thread is in an executing state.

See also: TSimpleIPCServer.ThreadError (1477), TSimpleIPCServer.StartServer (1474), TSimpleIPCServer.OnThreadError (1478), TSimpleIPCServer.Threaded (1479)

58.12.17 TSimpleIPCServer.ThreadError

Synopsis: Last thread error.

Declaration: `Property ThreadError : string`

Visibility: public

Access: Read

Description: `ThreadError` is the last error reported by the thread (or none if no error was caught).

See also: TSimpleIPCServer.ThreadExecuting (1477), TSimpleIPCServer.StartServer (1474), TSimpleIPCServer.OnThreadError (1478), TSimpleIPCServer.Threaded (1479)

58.12.18 TSimpleIPCServer.Global

Synopsis: Is the server reachable to all users or not.

Declaration: `Property Global : Boolean`

Visibility: published

Access: Read,Write

Description: `Global` indicates whether the server is reachable to all users (`True`) or if it is private to the current process (`False`). In the latter case, the unique channel endpoint identification may change: a unique identification of the current process is appended to the `ServerID` name.

See also: ServerID (1469), InstanceID (1477)

58.12.19 TSimpleIPCServer.OnMessage

Synopsis: Event triggered when a message arrives.

Declaration: `Property OnMessage : TNotifyEvent`

Visibility: published

Access: Read,Write

Description: `OnMessage` is called by ReadMessage (1475) when a message has been read. The actual message data can be retrieved with one of the StringMessage (1476), MsgData (1476) or MsgType (1476) properties.

See also: StringMessage (1476), MsgData (1476), MsgType (1476)

58.12.20 TSimpleIPCServer.OnMessageQueued

Synopsis: Event called when a new message has arrived.

Declaration: `Property OnMessageQueued : TNotifyEvent`

Visibility: published

Access: Read,Write

Description: `OnMessageQueued` is an event handler that is called whenever a new message is pushed on the queue.

See also: TSimpleIPCServer.PeekMessage (1475), TSimpleIPCServer.OnMessageError (1478)

58.12.21 TSimpleIPCServer.OnMessageError

Synopsis: Event called when a new message has arrived, and the queue is full.

Declaration: `Property OnMessageError : TMessageQueueEvent`

Visibility: published

Access: Read,Write

Description: `OnMessageError` is called whenever the message queue is full and a new message arrives on the server, andMaxAction (1468) is`ipcmoaError`.

58.12.22 TSimpleIPCServer.OnThreadError

Synopsis: Triggered when a thread reports an error.

Declaration: `Property OnThreadError : TNotifyEvent`

Visibility: published

Access: Read,Write

Description: `OnThreadError` is triggered when the server thread reports an error. The actual error message can be examined in TSimpleIPCServer.ThreadError (1477)

See also: TSimpleIPCServer.ThreadExecuting (1477), TSimpleIPCServer.StartServer (1474), TSimpleIPCServer.OnThreadError (1478), TSimpleIPCServer.ThreadError (1477), TSimpleIPCServer.Threaded (1479)

58.12.23 TSimpleIPCServer.MaxQueue

Synopsis: Maximum number of messages in the queue, 0 for unlimited.

Declaration: `Property MaxQueue : Integer`

Visibility: published

Access: Read,Write

Description: `MaxQueue` is the maximum number of messages in the queue. When this amount is zero, the amount of messages is unlimited.

When a new message is pushed, and the Count (1468) is equal to `MaxQueue`, the MaxAction (1479) property is examined to know what to do.

See also: MaxAction (1479)

58.12.24 TSimpleIPCServer.MaxAction

Synopsis: Action to take when the number of messages will exceed `MaxQueue`.

Declaration: `Property MaxAction : TIPCMessageOverflowAction`

Visibility: published

Access: Read,Write

Description: `MaxAction` determines what will happen if the number of messages on the queue equals MaxQueue (1479) and a new message is put in the queue during PeekMessage (1475):

Do nothing, just add the message.

Discard the oldest message.

Discard the new message.

See also: TSimpleIPCServer.MaxQueue (1479), TSimpleIPCServer.PeekMessage (1475), TIPCMessageOverflowAction (1458)

58.12.25 TSimpleIPCServer.Threaded

Synopsis: Is the server running threaded or not ?

Declaration: `Property Threaded : Boolean`

Visibility: published

Access: Read,Write

Description: `Threaded` indicates whether the server was started in threaded mode or not. It can be set before calling StartServer (1474). Trying to set it when the server is started will result in an error.

See also: StartServer (1474)

58.12.26 TSimpleIPCServer.ThreadTimeout

Synopsis: Timeout waiting for message.

Declaration: `Property ThreadTimeout : Integer`

Visibility: published

Access: Read,Write

Description: `ThreadTimeOut` is the time the thread will wait for messages between loop iterations, if the server is started with threading enabled.

When stopping the server, this is also the maximum time the server will be blocked when stopping, because it needs to wait for the thread to stop.

See also: TSimpleIPCServer.StartServer (1474), TSimpleIPCServer.StopServer (1475)

58.12.27 TSimpleIPCServer.SynchronizeEvents

Synopsis: Should events be run in the main thread ?

Declaration: `Property SynchronizeEvents : Boolean`

Visibility: published

Access: Read,Write

Description: `SynchronizeEvents` can be set to `True` to force execution of events in the main thread, when the server is running in threaded mode. If set to `False`, the events will be triggered in the thread responsible for checking messages. It is ignored when the server is not running threaded. It cannot be set when the server is already started.

See also: TSimpleIPCServer.Threaded (1479), StartServer (1474)

Chapter 59

Reference for unit 'singleinstance'

59.1 Used units

Table 59.1: Used units by unit 'singleinstance'

Name	Page
Classes	??
System	??
sysutils	??

59.2 Overview

`Singleinstance` contains the basic abstract definition of a class TBaseSingleInstance (1482) that, when instantiated, will make sure only 1 instance of your application will be running. It is an abstract instance, you need to instantiate a descendent such as can be found in the advancedsingleinstance (1481) unit.

In this document, the first started instance of an application is called the `Server`, any application started later is called a `Client` application.

The single instance functionality also provides a mechanism to communicate parameters from the client to the server. For example, an editor started a second time may wish to pass any filenames to be opened to the server instance.

59.3 Constants, types and variables

59.3.1 Types

```
TBaseSingleInstanceClass = Class of TBaseSingleInstance
```

`TBaseSingleInstanceClass` is the class of TBaseSingleInstance (1482). It is also the type of the DefaultSingleInstanceClass (1482) variable.

```
TSingleInstanceParamsEvent = procedure(Sender: TBaseSingleInstance
  ;
```

```
                                        Params: TStringList) of
  object
```

This callback type is the signature of the TBaseSingleInstance.OnServerReceivedParams (1485) event in case the server receives parameters from a newly started client.

Sender The `TBaseSingleInstance` which fired the event.

Params The client-provided parameters received by `TBaseSingleInstance`.

```
TSingleInstanceStart = (siServer,siClient,siNotResponding)
```

Table 59.2: Enumeration values for type TSingleInstanceStart

Value	Explanation
siClient	The current instance is the client (another instance was started earlier).
siNotResponding	The first instance is not responding.
siServer	The current instance is the server (first started).

`TSingleInstanceStart` is used to determine which kind of application the currently running application is.

siServer The current instance is the server (first started).

siClient The current instance is the client (another instance was started earlier).

siNotResponding The first instance is not responding.

59.3.2 Variables

```
DefaultSingleInstanceClass : TBaseSingleInstanceClass = Nil
```

`DefaultSingleInstanceClass` can be set to a class that implements the abstract methods of TBaseSingleInstance (1482). It can be used by applications to instantiate a class without referring directly to that class.

59.4 ESingleInstance

59.4.1 Description

`ESingleInstance` is used to report errors in the single instance implementation(s).

59.5 TBaseSingleInstance

59.5.1 Description

`TBaseSingleInstance` is the abstract parent for classes that allow you to control the instances if your application. You must never instantiate `TBaseSingleInstance` directly, instead, use for example the TAdvancedSingleInstance (167) class.

See also: TAdvancedSingleInstance (167), advancedsingleinstance (166)

59.5.2 Method overview

Page	Method	Description
1484	ClientPostParams	Send parameters to the server.
1483	Create	Create a new instance of the `TBaseSingleInstance` class.
1483	Destroy	Stop waiting and destroy the instance.
1484	ServerCheckMessages	Check for messages from a client application.
1483	Start	Check if another instance of the application is running.
1484	Stop	Stop the current instance.

59.5.3 Property overview

Page	Properties	Access	Description
1486	IsClient	r	Is this application the client instance ?
1486	IsServer	r	Is this application the server instance ?
1485	OnServerReceivedParams	rw	Event triggered during `ServerCheckMessages`.
1485	StartResult	r	Contains the result of the last `Start` call.
1485	TimeOutMessages	rw	Timeout when waiting for messages.
1485	TimeOutWaitForInstances	rw	Timeout when trying to contact server instance.

59.5.4 TBaseSingleInstance.Create

Synopsis: Create a new instance of the `TBaseSingleInstance` class.

Declaration:
```
constructor Create(aOwner: TComponent);  Override
```

Visibility: public

Description: `Create` calls the inherited constructor and then initializes some properties TimeOutMessages (1485) and TimeOutWaitForInstances (1485)

See also: TBaseSingleInstance.Destroy (1483), TimeOutMessages (1485), TimeOutWaitForInstances (1485)

59.5.5 TBaseSingleInstance.Destroy

Synopsis: Stop waiting and destroy the instance.

Declaration:
```
destructor Destroy;  Override
```

Visibility: public

Description: `Destroy` calls Stop (1484) and calls the inherited destructor.

See also: Stop (1484)

59.5.6 TBaseSingleInstance.Start

Synopsis: Check if another instance of the application is running.

Declaration:
```
function Start : TSingleInstanceStart;  Virtual;  Abstract
```

Visibility: public

Description: `Start` will check if another instance is running. It returns the result of the check, and the result is also stored in StartResult (1485)

siServerThe current instance is the server (first started).

siClientThe current instance is the client (another instance was started earlier).

siNotRespondingThe first instance is not responding.

See also: StartResult (1485), TSingleInstanceStart (1482)

59.5.7 TBaseSingleInstance.Stop

Synopsis: Stop the current instance.

Declaration: `procedure Stop; Virtual; Abstract`

Visibility: public

Description: `Stop` will disable the communication mechanism that ensures only a single instance is running. After it has been called in the server application, if another application calls Start (1483), then it will become the server (i.e. the result is `siServer`). If it is called in a client application, there is no effect.

See also: Start (1483)

59.5.8 TBaseSingleInstance.ServerCheckMessages

Synopsis: Check for messages from a client application.

Declaration: `procedure ServerCheckMessages; Virtual; Abstract`

Visibility: public

Description: `ServerCheckMessages` must be called at regular intervals by the server application to see if any client application sent messages (using TBaseSingleInstance.ClientPostParams (1484)). This is part of themechanism to communicate parameters from the client to the server. If there are any messages, then the OnServerReceivedParams (1485) event will be triggered once for any client that sent parameters.

See also: TBaseSingleInstance.ClientPostParams (1484)

59.5.9 TBaseSingleInstance.ClientPostParams

Synopsis: Send parameters to the server.

Declaration: `procedure ClientPostParams; Virtual; Abstract`

Visibility: public

Description: `ClientPostParams` must be called to send the client command-line parameters to the server application. The mechanism used to send these parameters depends on the used descendent of the TBaseSingleInstance (1482) class.

See also: TBaseSingleInstance.ServerCheckMessages (1484)

59.5.10 TBaseSingleInstance.TimeOutMessages

Synopsis: Timeout when waiting for messages.

Declaration: `Property TimeOutMessages : Integer`

Visibility: public

Access: Read,Write

Description: `TimeOutMessages` is the timeout (in milliseconds) used by ServerCheckMessages (1484) when checking for messages sent by client applications.

See also: ServerCheckMessages (1484), TimeOutWaitForInstances (1485)

59.5.11 TBaseSingleInstance.TimeOutWaitForInstances

Synopsis: Timeout when trying to contact server instance.

Declaration: `Property TimeOutWaitForInstances : Integer`

Visibility: public

Access: Read,Write

Description: `TimeOutMessages` is the timeout (in milliseconds) used by Start (1483) call when checking for another instance of the application.

See also: Start (1483), TimeOutMessages (1485)

59.5.12 TBaseSingleInstance.OnServerReceivedParams

Synopsis: Event triggered during `ServerCheckMessages`.

Declaration: `Property OnServerReceivedParams : TSingleInstanceParamsEvent`

Visibility: public

Access: Read,Write

Description: `OnServerReceivedParams` is the event triggered when ServerCheckMessages (1484) is called, and a client sent a message to the server. The `Params` parameter contains the commandline parameters that the client sent, one per string in the list.

59.5.13 TBaseSingleInstance.StartResult

Synopsis: Contains the result of the last `Start` call.

Declaration: `Property StartResult : TSingleInstanceStart`

Visibility: public

Access: Read

Description: `StartResult` contains the result of the last Start (1483) call.

See also: Start (1483)

59.5.14 TBaseSingleInstance.IsServer

Synopsis: Is this application the server instance ?

Declaration: `Property IsServer : Boolean`

Visibility: public

Access: Read

Description: `IsServer` is a convenience property, it is `True` if StartResult (1485) equals `siServer`.

See also: IsClient (1486), Start (1483), StartResult (1485)

59.5.15 TBaseSingleInstance.IsClient

Synopsis: Is this application the client instance ?

Declaration: `Property IsClient : Boolean`

Visibility: public

Access: Read

Description: `IsClient` is a convenience property, it is `True` if StartResult (1485) equals `siClient`.

See also: IsServer (1486), Start (1483), StartResult (1485)

Chapter 60

Reference for unit 'SQLDB'

60.1 Used units

Table 60.1: Used units by unit 'SQLDB'

Name	Page
BufDataset	213
Classes	??
DB	438
sqlscript	??
SQLTypes	1563
System	??
sysutils	??

60.2 Overview

The SQLDB unit defines four main classes to handle data in SQL based databases.

1. TSQLConnection (1514) represents the connection to the database. Here, properties pertaining to the connection (machine, database, user password) must be set. This is an abstract class, which should not be used directly. Per database type (mysql, firebird, postgres, oracle, sqlite) a descendent should be made and used.
2. TSQLQuery (1529) is a #fcl.db.TDataset (496) descendent which can be used to view and manipulate the result of an SQL select query. It can also be used to execute all kinds of SQL statements.
3. TSQLTransaction (1558) represents the transaction in which an SQL command is running. SQLDB supports multiple simultaneous transactions in a database connection. For databases that do not support this functionality natively, it is simulated by maintaining multiple connections to the database.
4. TSQLScript (1545) can be used when many SQL commands must be executed on a database, for example when creating a database.

There is also a unified way to retrieve schema information, and a registration for connector types. More information on how to use these components can be found in UsingSQLDB (1488).

60.3 Using SQLDB to access databases.

SQLDB can be used to connect to any SQL capable database. It allows to execute SQL statements on any supported database type in a uniform way, and allows to fetch and manipulate result sets (such as returned by a SELECT statement) using a standard TDataset (496) interface. SQLDB takes care that updates to the database are posted automatically to the database, in a cached manner.

When using SQLDB, 3 components are always needed:

1. A TSQLConnection (1514) descendent. This represents the connection to the database: the location of the database, and the username and password to authenticate the connection must be specified here. For each supported database type (Firebird, PostgreSQL, MySQL) there is a separate connection component. They all descend from `TSQLConnection`.
2. A TSQLTransaction (1558) component. SQLDB allows you to have multiple active but independent transactions in your application. (useful for instance in middle-tier applications). If the native database client library does not support this directly, it is emulated using multiple connections to the database.
3. A TSQLQuery (1529) component. This encapsulates an SQL statement. Any kind of SQL statement can be executed. The `TSQLQuery` component is a `TDataset` descendent: If the statement returns a result set, then it can be manipulated using the usual `TDataset` mechanisms.

The 3 components must be linked together: the connection must point to a default transaction (it is used to execute certain queries for metadata), the transaction component must point to a connection component. The TSQLQuery component must point to both a transaction and a database.

So in order to view the contents of a table, typically the procedure goes like this:

```
{$mode objfpc}{$h+}
uses sqldb, ibconnection;

Var
  C : TSQLConnection;
  T : TSQLTransaction;
  Q : TSQLQuery;

begin
  // Create a connection.
  C:=TIBConnection.Create(Nil);
  try
    // Set credentials.
    C.UserName:='MyUSER';
    C.Password:='Secret';
    C.DatabaseName:='/home/firebird/events.fb';
    // Create a transaction.
    T:=TSQLTransaction.Create(C);
    // Point to the database instance
    T.Database:=C;
    // Now we can open the database.
    C.Connected:=True;
    // Create a query to return data
    Q:=TSQLQuery.Create(C);
    // Point to database and transaction.
```

```
    Q.Database:=C;
    Q.Transaction:=T;
    // Set the SQL select statement
    Q.SQL.Text:='SELECT * FROM USERS';
    // And now use the standard TDataset methods.
    Q.Open;
    While not Q.EOF do
      begin
      Writeln(Q.FieldByName('U_NAME').AsString);
      Q.Next
      end;
    Q.Close;
  finally
    C.Free;
  end;
end.
```

The above code is quite simple. The connection type is `TIBConnection`, which is used for Firebird/Interbase databases. To connect to another database (for instance PostgreSQL), the exact same code could be used, but instead of a `TIBConnection`, a `TPQConnection` component must be used:

```
{$mode objfpc}{$h+}
uses sqldb, pqconnection;

Var
  C : TSQLConnection;
  T : TSQLTransaction;
  Q : TSQLQuery;

begin
  // Create a connection.
  C:=TPQConnection.Create(Nil);
```

The rest of the code remains identical.

The above code used an SQL SELECT statement and the `Open` method to fetch data from the database. Almost the same method applies when trying to execute other kinds of queries, such as DDL queries:

```
{$mode objfpc}{$h+}
uses sqldb, ibconnection;

Var
  C : TSQLConnection;
  T : TSQLTransaction;
  Q : TSQLQuery;

begin
  C:=TIBConnection.Create(Nil);
  try
    C.UserName:='MyUSER';
    C.Password:='Secret';
    C.DatabaseName:='/home/firebird/events.fb';
```

```
    T:=TSQLTransaction.Create(C);
    T.Database:=C;
    C.Connected:=True;
    Q:=TSQLQuery.Create(C);
    Q.Database:=C;
    Q.Transaction:=T;
    // Set the SQL statement. SQL is a tstrings instance.
    With Q.SQL do
      begin
      Add('CREATE TABLE USERS ( ');
      Add(' U_NAME VARCHAR(50), ');
      Add(' U_PASSWORD VARCHAR(50) ');
      Add(' ) ');
      end;
    // And now execute the query using ExecSQL
    // There is no result, so Open cannot be used.
    Q.ExecSQL;
    // Commit the transaction.
    T.Commit;
  finally
    C.Free;
  end;
end.
```

As can be seen from the above example, the setup is the same as in the case of fetching data. Note that TSQLQuery (1529) can only execute 1 SQL statement during ExecSQL. If many SQL statements must be executed, TSQLScript (1545) must be used.

There is much more to `TSQLQuery` than explained here: it can use parameters (see UsingParams (1492)) and it can automatically update the data that you edit in it (see UpdateSQLs (1491)).

See also: TSQLConnection (1514), TSQLTransaction (1558), TSQLQuery (1529), TSQLConnector (1525), TSQLScript (1545), UsingParams (1492), UpdateSQLs (1491)

60.4 Using the universal TSQLConnector type.

The normal procedure when using SQLDB is to use one of the TSQLConnection (1514) descendent components. When the database backend changes, another descendent of `TSQLConnection` must be used. When using a lot of different connection types and components, this may be confusing and a lot of work.

There is a universal connector component TSQLConnector (1525) which can connect to any database supported by SQLDB: it works as a proxy. Behind the scenes it uses a normal `TSQLConnection` descendent to do the real work. All this happens transparently to the user code, the universal connector acts and works like any normal connection component.

The type of database can be set in its ConnectorType (1526) property. By setting the `ConnectorType` property, the connector knows which `TSQLConnection` descendent must be created.

Each `TSQLConnection` descendent registers itself with a unique name in the initialization section of the unit implementing it: this is the name that should be specified in the `ConnectorType` of the universal connection. The list of available connections can be retrieved with the GetConnectionList (1501) call.

From this mechanism it follows that before a particular connection type can be used, its definition must be present in the list of connector types. This means that the unit of the connection type

(ibconnection, pqconnection etc.) must be included in the uses clause of the program file: if it is not included, the connection type will not be registered, and it will not be available for use in the universal connector.

The universal connector only exposes the properties common to all connection types (the ones in TSQLConnection). It does not expose properties for all the properties available in specific TSQLConnection descendents. This means that if connection-specific options must be used, they must be included in the Params (1525) property of the universal connector in the form Name=Value. When the actual connection instance is created, the connection-specific properties will be set from the specified parameters.

See also: TSQLConnection (1514), TSQLConnector (1525)

60.5 Retrieving Schema Information.

Schema Information (lists of available database objects) can be retrieved using some specialized calls in TSQLConnection (1514):

- TSQLConnection.GetTableNames (1517) retrieves a list of available tables. The system tables can be requested.
- TSQLConnection.GetProcedureNames (1517) retrieves a list of available stored procedures.
- TSQLConnection.GetFieldNames (1518) retrieves a list of fields for a given table.

These calls are pretty straightforward and need little explanation. A more versatile system is the schema info query: the TCustomSQLQuery.SetSchemaInfo (1508) method can be used to create a result set (dataset) with schema information. The parameter SchemaType determines the resulting information when the dataset is opened. The following information can be requested:

stTables Retrieves the list of user Tables in database. This is used internally by TSQLConnection.GetTableNames (1517).

stSysTables Retrieves the list of system Tables in database. This is used internally by TSQLConnection.GetTableNames (1517) when the system tables are requested

stProcedures Retrieves a list of stored procedures in database. This is used internally by TSQLConnection.GetProcedureNames (1517).

stColumns Retrieves the list of columns (fields) in a table. This is used internally by TSQLConnection.GetFieldNames (1518).

stProcedureParams This retrieves the parameters for a stored procedure.

stIndexes Retrieves the indexes for one or more tables. (currently not implemented)

stPackages Retrieves packages for databases that support them. (currently not implemented).

60.6 Automatic generation of update SQL statements.

SQLDB (more in particular, TSQLQuery (1529)) can automatically generate update statements for the data it fetches. To this end, it will scan the SQL statement and determine the main table in the query: this is the first table encountered in the FROM part of the SELECT statement.

For INSERT and UPDATE operations, the SQL statement will update/insert all fields that have pfInUpdate in their ProviderFlags property. Read-only fields will not be added to the SQL statement. Fields that are NULL will not be added to an insert query, which means that the database server will insert whatever is in the DEFAULT clause of the corresponding field definition.

The WHERE clause for update and delete statements consists of all fields with pfInKey in their ProviderFlags property. Depending on the value of the UpdateMode (1542) property, additional fields may be added to the WHERE clause:

upWhereKeyOnly No additional fields are added: only fields marked with pfInKey are used in the WHERE clause

upWhereChanged All fields whose value changed are added to the WHERE clause, using their old value.

upWhereAll All fields are added to the WHERE clause, using their old value.

In order to let SQLDB generate correct statements, it is important to set the ProviderFlags (573) properties correct for all fields.

In many cases, for example when only a single table is queried, and no AS field aliases are used, setting TSQLQuery.UsePrimaryKeyAsKey (1542) combined with UpdateMode equal to upWhereKeyOnly is sufficient.

If the automatically generated queries are not correct, it is possible to specify the SQL statements to be used in the UpdateSQL (1538), InsertSQL (1538) and DeleteSQL (1539) properties. The new field values should be specified using params with the same name as the field. The old field values should be specified using the OLD_ prefix to the field name. The following example demonstrates this:

```
INSERT INTO MYTABLE
  (MYFIELD,MYFIELD2)
VALUES
  (:MYFIELD,:MYFIELD2);

UPDATE MYTABLE SET
  MYFIELD=:MYFIELD
  MYFIELD2=:MYFIELD2
WHERE
  (MYFIELD=:OLD_MYFIELD);

DELETE FROM MYTABLE WHERE (MyField=:OLD_MYFIELD);
```

See also: UsingParams (1492), TSQLQuery (1529), UpdateSQL (1538), InsertSQL (1538), DeleteSQL (1538)

60.7 Using parameters.

SQLDB implements parameterized queries, simulating them if the native SQL client does not support parameterized queries. A parameterized query means that the SQL statement contains placeholders for actual values. The following is a typical example:

```
SELECT * FROM MyTable WHERE (id=:id)
```

The :id is a parameter with the name id. It does not contain a value yet. The value of the parameter will be specified separately. In SQLDB this happens through the TParams collection, where each

element of the collection is a named parameter, specified in the SQL statement. The value can be specified as follows:

```
Params.ParamByname('id').AsInteger:=123;
```

This will tell SQLDB that the parameter `id` is of type integer, and has value 123.

SQLDB uses parameters for 3 purposes:

1. When executing a query multiple times, simply with different values, this helps increase the speed if the server supports parameterized queries: the query must be prepared only once.
2. Master-Detail relationships between datasets can be established based on a parameterized detail query: the value of the parameters in the detail query is automatically obtained from fields with the same names in the master dataset. As the user scrolls through the master dataset, the detail dataset is refreshed with the new values of the params.
3. Updating of data in the database happens through parameterized update/delete/insert statements: the TSQLQuery.UpdateSQL (1538), TSQLQuery.DeleteSQL (1539), TSQLQuery.InsertSQL (1538) properties of TSQLQuery (1529) must contain parameterized queries.

An additional advantage of using parameters is that they help to avoid SQL injection: by specifying a parameter type and value, SQLDB will automatically check whether the value is of the correct type, and will apply proper quoting when the native engine does not support parameters directly.

See also: TSQLQuery.Params (1540), UpdateSQLs (1491)

60.8 Constants, types and variables

60.8.1 Constants

```
DefaultMacroChar = '%'
```

`DefaultMacroChar` is the default macro delimiter to use in TSQLQuery (1529)

```
DefaultSQLFormatSettings : TFormatSettings = (CurrencyFormat: 1; NegCurrFormat
  : 5; ThousandSeparator: #0; DecimalSeparator: '.'; CurrencyDecimals
  : 2; DateSeparator: '-'; TimeSeparator: ':'; ListSeparator: ' '; CurrencyString
  : '$'; ShortDateFormat: 'yyyy-mm-dd'; LongDateFormat: ''; TimeAMString
  : ''; TimePMString: ''; ShortTimeFormat: 'hh:nn:ss'; LongTimeFormat
  : 'hh:nn:ss.zzz'; ShortMonthNames: ('', '', '', '', '', '', '', ''
  , '', '', '', ''); LongMonthNames: ('', '', '', '', '', '', '', ''
  , '', '', '', ''); ShortDayNames: ('', '', '', '', '', '', ''); LongDayNames
  : ('', '', '', '', '', '', ''); TwoDigitYearCenturyWindow: 50)
```

`DefaultSQLFormatSettings` contains the default settings used when formatting date/time and other special values in Update SQL statements generated by the various TSQLConnection (1514) descendents.

```
detActualSQL = sqltypes.detActualSQL
```

Alias for `sqltypes.detActualSQL`.

```
detCommit = sqltypes.detCommit
```

Alias for `sqltypes.detCommit`.

```
detCustom = sqltypes.detCustom
```

Alias for `sqltypes.detCustom`.

```
detExecute = sqltypes.detExecute
```

Alias for `sqltypes.detExecute`.

```
detFetch = sqltypes.detFetch
```

Alias for `sqltypes.detFetch`.

```
detParamValue = sqltypes.detParamValue
```

Alias for `sqltypes.detParamValue`.

```
detPrepare = sqltypes.detPrepare
```

Alias for `sqltypes.detPrepare`.

```
detRollBack = sqltypes.detRollBack
```

Alias for `sqltypes.detRollBack`.

```
DoubleQuotes : TQuoteChars = ('"', '"')
```

`DoubleQuotes` is the set of delimiters used when using double quotes for string literals.

```
LogAllEvents = [detCustom, detPrepare, detExecute, detFetch, detCommit
  , detRollBack]
```

`LogAllEvents` is a constant that contains the full set of available event types. It can be used to set TSQLConnection.LogEvents (1523).

```
LogAllEventsExtra = [detCustom, detPrepare, detExecute, detFetch,
  detCommit, detRollBack, detParamValue, detActualSQL]
```

`LogAllEventsExtra` lists all possible even types that can be reported using the connection logging mechanism.

```
SingleQuotes : TQuoteChars = ('''', '''')
```

`SingleQuotes` is the set of delimiters used when using single quotes for string literals.

```
StatementTokens : Array[TStatementType] of string = ('(unknown)',
  'select', 'insert', 'update', 'delete', 'create', 'get', 'put', 'execute'
  , 'start', 'commit', 'rollback', '?')
```

`StatementTokens` contains an array of string tokens that are used to detect the type of statement, usually the first SQL keyword of the token. The presence of this token in the SQL statement determines the kind of token.

```
stColumns = sqltypes.stColumns
```

Alias for `sqltypes.stColumns`.

```
stCommit = sqltypes.stCommit
```

Alias for `sqltypes.stCommit`.

```
stDDL = sqltypes.stDDL
```

Alias for `sqltypes.stDDL`.

```
stDelete = sqltypes.stDelete
```

Alias for `sqltypes.stDelete`.

```
stExecProcedure = sqltypes.stExecProcedure
```

Alias for `sqltypes.stExecProcedure`.

```
stGetSegment = sqltypes.stGetSegment
```

Alias for `sqltypes.stGetSegment`.

```
stIndexes = sqltypes.stIndexes
```

Alias for `sqltypes.stIndexes`.

```
stInsert = sqltypes.stInsert
```

Alias for `sqltypes.stInsert`.

```
stNoSchema = sqltypes.stNoSchema
```

Alias for `sqltypes.stUnknown`.

```
stPackages = sqltypes.stPackages
```

Alias for `sqltypes.stPackages`.

```
stProcedureParams = sqltypes.stProcedureParams
```

Alias for `sqltypes.stProcedureParams`.

```
stProcedures = sqltypes.stProcedures
```

Alias for `sqltypes.stProcedures`.

```
stPutSegment = sqltypes.stPutSegment
```

Alias for `sqltypes.stPutSegment`.

```
stRollback = sqltypes.stRollback
```

Alias for `sqltypes.stRollback`.

```
stSchemata = sqltypes.stSchemata
```

`stSchemata` is a convenience alias for #fcl.sqlTypes.stSchemata (1563).

```
stSelect = sqltypes.stSelect
```

Alias for `sqltypes.stSelect`.

```
stSelectForUpd = sqltypes.stSelectForUpd
```

Alias for `sqltypes.stSelectForUpd`.

```
stSequences = sqltypes.stSequences
```

Alias for `sqltypes.stSequences`.

```
stStartTrans = sqltypes.stStartTrans
```

Alias for `sqltypes.stStartTrans`.

```
stSysTables = sqltypes.stSysTables
```

Alias for `sqltypes.stSysTables`.

```
stTables = sqltypes.stTables
```

Alias for `sqltypes.stTables`.

```
stUnknown = sqltypes.stUnknown
```

Alias for `sqltypes.stUnknown`.

```
stUpdate = sqltypes.stUpdate
```

Alias for `sqltypes.stUpdate`.

```
TSchemaObjectNames : Array[TSchemaType] of string = ('???', 'table_name'
  , '???', 'procedure_name', 'column_name', 'param_name', 'index_name'
  , 'package_name', 'schema_name', 'sequence')
```

Names of the various types of objects.

60.8.2 Types

```
TCommitRollbackAction = (caNone,caCommit,caCommitRetaining,caRollback
  ,
                          caRollbackRetaining)
```

Table 60.2: Enumeration values for type TCommitRollbackAction

Value	Explanation
caCommit	Commit transaction.
caCommitRetaining	Commit transaction, retaining transaction context.
caNone	Do nothing.
caRollback	Rollback transaction.
caRollbackRetaining	Rollback transaction, retaining transaction context.

`TCommitRollbackAction` is currently unused in SQLDB.

```
TConnectionDefClass = Class of TConnectionDef
```

`TConnectionDefClass` is used in the RegisterConnection (1501) call to register a new TConnectionDef (1503) instance.

```
TConnInfoType = (citAll,citServerType,citServerVersion,
  citServerVersionString,citClientName,citClientVersion)
```

Table 60.3: Enumeration values for type TConnInfoType

Value	Explanation
citAll	All connection information.
citClientName	Client library name.
citClientVersion	Client library version.
citServerType	Server type description.
citServerVersion	Server version as an integer number.
citServerVersionString	Server version as a string.

Connection information to be retrieved.

```
TConnOption = (sqSupportParams,sqSupportEmptyDatabaseName,sqEscapeSlash
  ,
               sqEscapeRepeat,sqImplicitTransaction,sqLastInsertID
  ,
               sqSupportReturning,sqSequences,sqCommitEndsPrepared
  ,
               sqRollbackEndsPrepared)
```

Table 60.4: Enumeration values for type TConnOption

Value	Explanation
sqCommitEndsPrepared	
sqEscapeRepeat	Escapes in string literals are done by repeating the character.
sqEscapeSlash	Escapes in string literals are done with backslash characters.
sqImplicitTransaction	Does the connection support implicit transaction management.
sqLastInsertID	Does the connection support getting the ID for the last insert operation.
sqRollbackEndsPrepared	
sqSequences	Are sequences supported.
sqSupportEmptyDatabaseName	Does the connection allow empty database names ?
sqSupportParams	The connection type has native support for parameters.
sqSupportReturning	The connection type supports INSERT/UPDATE with RETURNING clause.

This type describes some of the option that a particular connection type supports.

```
TConnOptions = Set of TConnOption
```

`TConnOptions` describes the full set of options defined by a database.

```
TDBEventType = sqltypes.TDBEventType
```

`TDBEventType` describes the type of a database event message as generated by TSQLConnection (1514) through the TSQLConnection.OnLog (1522) event.

```
TDBEventTypes = sqltypes.TDBEventTypes
```

`TDBEventTypes` is a set of TDBEventType (1498) values, which is used to filter the set of event messages that should be sent. The TSQLConnection.LogEvents (1523) property determines which events a particular connection will send.

```
TDBLogNotifyEvent = procedure(Sender: TSQLConnection;
  EventType: TDBEventType;
  const Msg: string) of object
```

`TDBLogNotifyEvent` is the prototype for the TSQLConnection.OnLog (1522) event handler and for the global GlobalDBLogHook (1500) event handling hook. `Sender` will contain the TSQLConnection (1514) instance that caused the event, `EventType` will contain the event type, and `Msg` will contain the actual message: the content depends on the type of the message.

```
TLibraryLoadFunction = function(const S: AnsiString) : Integer
```

`TLibraryLoadFunction` is the function prototype for dynamically loading a library when the universal connection component is used. It receives the name of the library to load (`S`), and should return `True` if the library was successfully loaded. It is used in the connection definition.

```
TLibraryUnLoadFunction = procedure
```

`TLibraryUnLoadFunction` is the function prototype for dynamically unloading a library when the universal connection component is used. It has no parameters, and should simply unload the library loaded with TLibraryLoadFunction (1498)

```
TQuoteChars = sqltypes.TQuoteChars
```

TQuoteChars is an array of characters that describes the used delimiters for string values.

```
TRowsCount = LargeInt
```

A type to contain a result row count.

```
TSchemaType = sqltypes.TSchemaType
```

TSchemaType describes which schema information to retrieve in the TCustomSQLQuery.SetSchemaInfo (1508) call. Depending on its value, the result set of the dataset will have different fields, describing the requested schema data. The result data will always have the same structure.

```
TSQLConnectionClass = Class of TSQLConnection
```

TSQLConnectionClass is used when registering a new connection type for use in the universal connector TSQLConnector.ConnectorType (1526)

```
TSQLConnectionOption = (scoExplicitConnect,
  scoApplyUpdatesChecksRowsAffected)
```

Table 60.5: Enumeration values for type TSQLConnectionOption

Value	Explanation
scoApplyUpdatesChecksRowsAffected	ApplyUpdates will check that the RowsAffected is 1 after an update.
scoExplicitConnect	Require explicit connection to the database (default is implicit).

TSQLConnectionOption enumerates several options that can be set for TSQLConnection (1514) instances using TSQLConnection.Options (1523)

```
TSQLConnectionOptions = Set of TSQLConnectionOption
```

Set of TSQLConnectionOption.

```
TSQLQueryOption = (sqoKeepOpenOnCommit,sqoAutoApplyUpdates,
  sqoAutoCommit,sqoCancelUpdatesOnRefresh,
  sqoRefreshUsingSelect,sqoNoCloseOnSQLChange)
```

Table 60.6: Enumeration values for type TSQLQueryOption

Value	Explanation
sqoAutoApplyUpdates	Call ApplyUpdates on Post or Delete.
sqoAutoCommit	Call commit after every ApplyUpdates or ExecSQL.
sqoCancelUpdatesOnRefresh	Cancel any pending updates when refresh is called.
sqoKeepOpenOnCommit	Keep the dataset open after the query was committed (will fetch all records).
sqoNoCloseOnSQLChange	
sqoRefreshUsingSelect	Force a refresh using the provided select instead of using RETURNING clause.

TSQLQueryOption enumerates several options available to control the behaviour of an TSQLQuery (1529) instance.

```
TSQLQueryOptions = Set of TSQLQueryOption
```

TSQLQueryOptions is the type of the TSQLQuery.Options (1540) property.

```
TSQLSequenceApplyEvent = (saeOnNewRecord,saeOnPost)
```

Table 60.7: Enumeration values for type TSQLSequenceApplyEvent

Value	Explanation
saeOnNewRecord	Fetch an ID when a new record is appended to a dataset.
saeOnPost	Fetch an ID when a new record is posted in the dataset.

TSQLSequenceApplyEvent enumerates the moments when a new ID must be fetched for a sequence field.

```
TSQLTransactionOption = (stoUseImplicit,stoExplicitStart)
```

Table 60.8: Enumeration values for type TSQLTransactionOption

Value	Explanation
stoExplicitStart	Require explicit start of transactions by TSQLQuery.
stoUseImplicit	Use implicit transaction control if the engine allows it.

TSQLTransactionOption enumerates several options that can be used to control the transaction behaviour of TSQLTransaction (1558).

```
TSQLTransactionOptions = Set of TSQLTransactionOption
```

TSQLTransactionOptions is the property type of TSQLTransaction.Options (1562).

```
TStatementType = sqltypes.TStatementType
```

TStatementType describes the kind of SQL statement that was entered in the SQL property of a TSQLQuery (1529) component.

60.8.3 Variables

```
GlobalDBLogHook : TDBLogNotifyEvent
```

GlobalDBLogHook can be set in addition to local TSQLConnection.Onlog (1522) event handlers. All connections will report events through this global event handler in addition to their OnLog event handlers. The global log event handler can be set only once, so when setting the handler, it is important to set up chaining: saving the previous value, and calling the old handler (if it was set) in the new handler.

60.9 Procedures and functions

60.9.1 GetConnectionDef

Synopsis: Search for a connection definition by name.

Declaration: `function GetConnectionDef(const ConnectorName: string) : TConnectionDef`

Visibility: default

Description: `GetConnectionDef` will search in the list of connection type definitions, and will return the one definition with the name that matches `ConnectorName`. The search is case insensitive.

If no definition is found, `Nil` is returned.

See also: RegisterConnection (1501), TConnectionDef (1503), TConnectionDef.TypeName (1503)

60.9.2 GetConnectionList

Synopsis: Return a list of connection definition names.

Declaration: `procedure GetConnectionList(List: TStrings)`

Visibility: default

Description: `GetConnectionList` clears `List` and fills it with the list of currently known connection type names, as registered with RegisterConnection (1501). The names are the names as returned by TConnectionDef.TypeName (1503)

See also: RegisterConnection (1501), TConnectionDef.TypeName (1503)

60.9.3 RegisterConnection

Synopsis: Register a new connection type for use in the universal connector.

Declaration: `procedure RegisterConnection(Def: TConnectionDefClass)`

Visibility: default

Description: `RegisterConnection` must be called with a class pointer to a TConnectionDef (1503) descendent to register the connection type described in the TConnectionDef (1503) descendent. The connection type is registered with the name as returned by TConnectionDef.TypeName (1503).

The various connection types distributed by Free Pascal automatically call `RegisterConnection` from the `initialization` section of their unit, so simply including the unit with a particular connection type is enough to register it.

Connection types registered with this call can be unregistered with UnRegisterConnection (1502).

Errors: if `Def` is `Nil`, access violations will occur.

See also: TConnectionDef (1503), UnRegisterConnection (1502)

60.9.4 UnRegisterConnection

Synopsis: Unregister a registered connection type.

Declaration:
```
procedure UnRegisterConnection(Def: TConnectionDefClass)
procedure UnRegisterConnection(const ConnectionName: string)
```

Visibility: default

Description: `UnRegisterConnection` will unregister the connection `Def`. If a connection with `ConnectionName` or with name as returned by the TypeName (1503) method from `Def` was previously registered, it will be removed from the list of registered connection types.

Errors: if `Def` is `Nil`, access violations will occur.

See also: TConnectionDef (1503), RegisterConnection (1501)

60.10 TSQLStatementInfo

```
TSQLStatementInfo = record
  StatementType : TStatementType;
  TableName
  : string;
  Updateable : Boolean;
  WhereStartPos : Integer;
  WhereStopPos
  : Integer;
end
```

`TSQLStatementInfo` is a record used to describe an SQL statement. It is used internally by the TSQLStatement (1554) and TSQLQuery (1529) objects to analyse SQL statements.

It is used to be able to modify the SQL statement (for additional filtering) or to determine the table to update when applying dataset updates to the database.

60.11 ESQLDatabaseError

60.11.1 Description

`ESQLDatabaseError` is raised by SQLDB routines if the underlying engine raises an error. The error code returned by the engine is contained in ESQLDatabaseError.ErrorCode (**??**), and an Ansi SQL compliant SQL state can be passed in ESQLDatabaseError.SQLState (**??**)

See also: db.EDatabaseError (1487), ESQLDatabaseError.SQLState (**??**), ESQLDatabaseError.ErrorCode (**??**)

60.11.2 Method overview

Page	Method	Description
1502	CreateFmt	Create a new instance of `ESQLDatabaseError`.

60.11.3 ESQLDatabaseError.CreateFmt

Synopsis: Create a new instance of `ESQLDatabaseError`.

Declaration:
```
constructor CreateFmt(const Fmt: string; const Args: Array of const;
                      Comp: TComponent; AErrorCode: Integer;
                      const ASQLState: string);  Overload
```

Visibility: public

Description: `CreateFmt` is overloaded in `ESQLDatabaseError` to be able to specify the ErrorCode (**??**) and SQLState (**??**).

See also: ESQLDatabaseError.ErrorCode (**??**), ESQLDatabaseError.SQLState (**??**)

60.12 TConnectionDef

60.12.1 Description

`TConnectionDef` is an abstract class. When registering a new connection type for use in the universal connector, a descendent of this class must be made and registered using RegisterConnection (1501). A descendent class should override at least the TConnectionDef.TypeName (1503) and TConnectionDef.ConnectionClass (1504) methods to return the specific name and connection class to use.

See also: TConnectionDef.TypeName (1503), TConnectionDef.ConnectionClass (1504), RegisterConnection (1501)

60.12.2 Method overview

Page	Method	Description
1505	ApplyParams	Apply parameters to an instance of `TSQLConnection`.
1504	ConnectionClass	Class to instantiate when this connection is requested.
1504	DefaultLibraryName	Default library name.
1504	Description	A descriptive text for this connection type.
1505	LoadedLibraryName	Currently loaded library.
1504	LoadFunction	Return a function to call when the client library must be loaded.
1503	TypeName	Name of the connection type.
1505	UnLoadFunction	Return a function to call when the client library must be unloaded.

60.12.3 TConnectionDef.TypeName

Synopsis: Name of the connection type.

Declaration:
```
class function TypeName : string;  Virtual
```

Visibility: default

Description: `TypeName` is overridden by descendent classes to return the unique name for this connection type. It is what the TSQLConnector.ConnectorType (1526) property should be set to select this connection type for the universal connection, and is the name that the GetConnectionDef (1501) call will use when looking for a connection type. It must be overridden by descendents of `TConnectionDef`.

This name is also returned in the list returned by GetConnectionList (1501)

This name can be an arbitrary name, no restrictions on the allowed characters exist.

See also: TSQLConnector.ConnectorType (1526), GetConnectionDef (1501), GetConnectionList (1501), TConnectionDef.ConnectionClass (1504)

60.12.4 TConnectionDef.ConnectionClass

Synopsis: Class to instantiate when this connection is requested.

Declaration: `class function ConnectionClass : TSQLConnectionClass; Virtual`

Visibility: default

Description: `ConnectionClass` should return the connection class to use when a connection of this type is requested. It must be overridden by descendents of `TConnectionDef`.

It may not be `Nil`.

See also: TConnectionDef.TypeName (1503)

60.12.5 TConnectionDef.Description

Synopsis: A descriptive text for this connection type.

Declaration: `class function Description : string; Virtual`

Visibility: default

Description: `Description` should return a descriptive text for this connection type. It is used for display purposes only, so ideally it should be a one-liner. It can be used to provide more information about the particulars of the connection type.

See also: TConnectionDef.TypeName (1503)

60.12.6 TConnectionDef.DefaultLibraryName

Synopsis: Default library name.

Declaration: `class function DefaultLibraryName : string; Virtual`

Visibility: default

Description: `DefaultLibraryName` should be set to the default library name for the connection. This can be used to let SQLDB automatically load the library needed when a connection of this type is requested.

See also: TLibraryLoadFunction (1498), TConnectionDef (1503), TLibraryUnLoadFunction (1498)

60.12.7 TConnectionDef.LoadFunction

Synopsis: Return a function to call when the client library must be loaded.

Declaration: `class function LoadFunction : TLibraryLoadFunction; Virtual`

Visibility: default

Description: `LoadFunction` must return the function that will be called when the client library for this connection type must be loaded. This method must be overridden by descendent classes to return a function that will correctly load the client library when a connection of this type is used.

See also: TLibraryLoadFunction (1498), TConnectionDef.UnLoadFunction (1505), TConnectionDef.DefaultLibraryName (1504), TConnectionDef.LoadedLibraryName (1505)

60.12.8 TConnectionDef.UnLoadFunction

Synopsis: Return a function to call when the client library must be unloaded.

Declaration:
```
class function UnLoadFunction : TLibraryUnLoadFunction;  Virtual
```

Visibility: default

Description: `UnLoadFunction` must return the function that will be called when the client library for this connection type must be unloaded. This method must be overridden by descendent classes to return a function that will correctly unload the client library when a connection of this type is no longer used.

See also: TLibraryUnLoadFunction (1498), TConnectionDef.LoadFunction (1504), TConnectionDef.DefaultLibraryName (1504), TConnectionDef.LoadedLibraryName (1505)

60.12.9 TConnectionDef.LoadedLibraryName

Synopsis: Currently loaded library.

Declaration:
```
class function LoadedLibraryName : string;  Virtual
```

Visibility: default

Description: `LoadedLibraryName` must be overridden by descendents to return the filename of the currently loaded client library for this connection type. If no library is loaded, an empty string must be returned.

See also: TLibraryLoadFunction (1498), TLibraryUnLoadFunction (1498), TConnectionDef.LoadFunction (1504), TConnectionDef.UnLoadFunction (1505), TConnectionDef.DefaultLibraryName (1504)

60.12.10 TConnectionDef.ApplyParams

Synopsis: Apply parameters to an instance of `TSQLConnection`.

Declaration:
```
procedure ApplyParams(Params: TStrings; AConnection: TSQLConnection)
                          ;  Virtual
```

Visibility: default

Description: `ApplyParams` must be overridden to apply any params specified in the `Params` argument to the TSQLConnection (1514) descendent in `AConnection`. It can be used to convert `Name=Value` pairs to properties of the actual connection instance.

When called, `AConnection` is guaranteed to be of the same type as returned by TConnectionDef.ConnectionClass (1504). `Params` contains the contents of the TSQLConnection.Params (1525) property of the connector.

See also: TSQLConnection.Params (1525)

60.13 TCustomSQLQuery

60.13.1 Description

`TCustomSQLQuery` encapsulates a SQL statement: it implements all the necessary #fcl.db.TDataset (496) functionality to be able to handle a result set. It can also be used to execute SQL statements that do not return data, using the ExecSQL (1507) method.

Do not instantiate a `TCustomSQLQuery` class directly, instead use the TSQLQuery (1529) descendent.

See also: TSQLQuery (1529)

60.13.2 Method overview

Page	Method	Description
1509	ApplyUpdates	Apply updates and check result.
1506	Create	Create a new instance of `TCustomSQLQuery`.
1509	Delete	Delete and optionally apply updates.
1506	Destroy	Destroy instance of `TCustomSQLQuery`.
1507	ExecSQL	Execute a SQL statement that does not return a result set.
1509	MacroByName	Convenience for Macros.ParamByName.
1508	ParamByName	Return parameter by name.
1509	Post	Post pending changes and optionally apply updates.
1507	Prepare	Prepare a query for execution.
1508	RowsAffected	Return the number of rows (records) affected by the last DML/DDL statement.
1508	SetSchemaInfo	SetSchemaInfo prepares the dataset to retrieve schema info.
1507	UnPrepare	Unprepare a prepared query.

60.13.3 Property overview

Page	Properties	Access	Description
1510	Prepared	r	Is the query prepared ?
1510	SQLConnection	rw	Database as TSQLConnection.
1510	SQLTransaction	rw	Transaction as TSQLTransaction.

60.13.4 TCustomSQLQuery.Create

Synopsis: Create a new instance of `TCustomSQLQuery`.

Declaration:
```
constructor Create(AOwner: TComponent);  Override
```

Visibility: public

Description: `Create` allocates a new instance on the heap and will allocate all resources for the SQL statement. After this it calls the inherited constructor.

Errors: If not enough memory is available, an exception will be raised.

See also: TCustomSQLQuery.Destroy (1506)

60.13.5 TCustomSQLQuery.Destroy

Synopsis: Destroy instance of `TCustomSQLQuery`.

Declaration:
```
destructor Destroy;  Override
```

Visibility: public

Description: `Destroy` cleans up the instance, closing the dataset and freeing all allocated resources.

See also: TCustomSQLQuery.Create (1506)

60.13.6 TCustomSQLQuery.Prepare

Synopsis: Prepare a query for execution.

Declaration: `procedure Prepare; Virtual`

Visibility: public

Description: `Prepare` will prepare the SQL for execution. It will open the database connection if it was not yet open, and will start a transaction if none was started yet. It will then determine the statement type. Finally, it will pass the statement on to the database engine if it supports preparing of queries.

Strictly speaking, it is not necessary to call prepare, the component will prepare the statement whenever it is necessary. If a query will be executed repeatedly, it is good practice to prepare it once before starting to execute it. This will speed up execution, since resources must be allocated only once.

Errors: If the SQL server cannot prepare the statement, an exception will be raised.

See also: TSQLQuery.StatementType (1532), TCustomSQLQuery.UnPrepare (1507), TCustomSQLQuery.ExecSQL (1507)

60.13.7 TCustomSQLQuery.UnPrepare

Synopsis: Unprepare a prepared query.

Declaration: `procedure UnPrepare; Virtual`

Visibility: public

Description: `Unprepare` will unprepare a prepared query. This means that server resources for this statement are deallocated. After a query was unprepared, any `ExecSQL` or `Open` command will prepare the SQL statement again.

Several actions will unprepare the statement: Setting the TSQLQuery.SQL (1537) property, setting the `Transaction` property or setting the `Database` property will automatically call `UnPrepare`. Closing the dataset will also unprepare the query.

Errors: If the SQL server cannot unprepare the statement, an exception may be raised.

See also: TSQLQuery.StatementType (1532), TCustomSQLQuery.Prepare (1507), TCustomSQLQuery.ExecSQL (1507)

60.13.8 TCustomSQLQuery.ExecSQL

Synopsis: Execute a SQL statement that does not return a result set.

Declaration: `procedure ExecSQL; Virtual`

Visibility: public

Description: `ExecSQL` will execute the statement in TSQLQuery.SQL (1537), preparing the statement if necessary. It cannot be used to get results from the database (such as returned by a `SELECT` statement): for this, the Open (515) method must be used.

The SQL property should be a single SQL command. To execute multiple SQL statements, use the TSQLScript (1545) component instead.

If the statement is a `DML` statement, the number of deleted/updated/inserted rows can be determined using TCustomSQLQuery.RowsAffected (1508).

The `Database` and `Transaction` properties must be assigned before calling `ExecSQL`. Executing an empty SQL statement is also an error.

Errors: If the server reports an error, an exception will be raised.

See also: TCustomSQLQuery.RowsAffected (1508), TDataset.Open (515)

60.13.9 TCustomSQLQuery.SetSchemaInfo

Synopsis: SetSchemaInfo prepares the dataset to retrieve schema info.

Declaration:
```
procedure SetSchemaInfo(ASchemaType: TSchemaType;
                        ASchemaObjectName: string;
                        ASchemaPattern: string);  Virtual
```

Visibility: public

Description: `SetSchemaInfo` will prepare the dataset to retrieve schema information from the connection, and represents the schema info as a dataset.

`SetSchemaInfo` is used internally to prepare a query to retrieve schema information from a connection. It will store the 3 passed parameters, which are then used in the ParseSQL and Prepare stages to optimize the allocated resources. setting the schema type to anything other than `stNoSchema` will also set (or mimic) the SQL statement as soon as the query is prepared. For connection types that support this, the SQL statement is then set to whatever statement the database connection supports to retrieve schema information.

This is used internally by TSQLConnection.GetTableNames (1517) and TSQLConnection.GetProcedureNames (1517) to get the necessary schema information from the database.

See also: TSQLConnection.GetTableNames (1517), TSQLConnection.GetProcedureNames (1517), RetrievingSchemaInformation (1491)

60.13.10 TCustomSQLQuery.RowsAffected

Synopsis: Return the number of rows (records) affected by the last DML/DDL statement.

Declaration:
```
function RowsAffected : TRowsCount;  Virtual
```

Visibility: public

Description: `RowsAffected` returns the number of rows affected by the last statement executed using ExecSQL (1507).

Errors: If the connection or database type does not support returning this number, -1 is returned. If the query is not connected to a database, -1 is returned.

See also: TCustomSQLQuery.ExecSQL (1507), TSQLConnection (1514)

60.13.11 TCustomSQLQuery.ParamByName

Synopsis: Return parameter by name.

Declaration:
```
function ParamByName(const AParamName: string) : TParam
```

Visibility: public

Description: `ParamByName` is a shortcut for Params.ParamByName (634). The 2 following pieces of code are completely equivalent:

```
Qry.ParamByName('id').AsInteger:=123;
```

and

```
Qry.Params.ParamByName('id').AsInteger:=123;
```

See also: Params.ParamByName (634), TSQLQuery.Params (1540)

60.13.12 TCustomSQLQuery.MacroByName

Synopsis: Convenience for Macros.ParamByName.

Declaration: `function MacroByName(const AParamName: string) : TParam`

Visibility: public

Description: `MacroByName` checks Macros (1505) for the macro named `aParamName` and returns the requested macro.

Errors: If no macro is found, an exception is raised.

See also: #fcl.db.TParams.ParamByName (634)

60.13.13 TCustomSQLQuery.ApplyUpdates

Synopsis: Apply updates and check result.

Declaration: `procedure ApplyUpdates(MaxErrors: Integer); Override; Overload`

Visibility: public

Description: `ApplyUpdates` is overridden in TCustomSQLQuery (1505) to check the result of the update (using RowsAffected (1508)).

See also: TSQLQueryOptions (1500), TSQLQuery.Options (1540)

60.13.14 TCustomSQLQuery.Post

Synopsis: Post pending changes and optionally apply updates.

Declaration: `procedure Post; Override`

Visibility: public

Description: `Post` is overridden from DB.TDataset.Post (1487) to implement the auto-applyupdates mechanism: if TSQLQuery.Options (1540) contains `sqoAutoApplyUpdates`, then `ApplyUpdates` is called as the last step of the `Post` operation.

See also: TSQLQuery.Options (1540), TCustomSQLQuery.ApplyUpdates (1509)

60.13.15 TCustomSQLQuery.Delete

Synopsis: Delete and optionally apply updates.

Declaration: `procedure Delete; Override`

Visibility: public

Description: `Delete` is overridden from DB.TDataset.Delete (1487) to implement the auto-applyupdates mechanism: if TSQLQuery.Options (1540) contains `sqoAutoApplyUpdates`, then `ApplyUpdates` is called as the last step of the `Post` operation.

See also: TSQLQuery.Options (1540), TCustomSQLQuery.ApplyUpdates (1509)

60.13.16 TCustomSQLQuery.Prepared

Synopsis: Is the query prepared ?

Declaration: `Property Prepared : Boolean`

Visibility: public

Access: Read

Description: `Prepared` is true if Prepare (1507) was called for this query, and an UnPrepare (1507) was not done after that (take care: several actions call `UnPrepare` implicitly). Initially, `Prepared` will be `False`. Calling `Prepare` if the query was already prepared has no effect.

See also: TCustomSQLQuery.Prepare (1507), TCustomSQLQuery.UnPrepare (1507)

60.13.17 TCustomSQLQuery.SQLConnection

Synopsis: Database as TSQLConnection.

Declaration: `Property SQLConnection : TSQLConnection`

Visibility: public

Access: Read,Write

Description: `SQLConnection` equals the `Database` property, but typecasted as a TSQLConnection (1514) descendent.

See also: TSQLConnection (1514), TCustomSQLQuery.SQLTransaction (1510)

60.13.18 TCustomSQLQuery.SQLTransaction

Synopsis: Transaction as TSQLTransaction.

Declaration: `Property SQLTransaction : TSQLTransaction`

Visibility: public

Access: Read,Write

Description: `SQLTransaction` equals the `Transaction` property, but typecasted as a TSQLTransaction (1558) descendent.

See also: TSQLConnection (1514), TCustomSQLQuery.SQLConnection (1510)

60.14 TCustomSQLStatement

60.14.1 Description

`TCustomSQLStatement` is a light-weight object that can be used to execute SQL statements on a database. It does not support result sets, and has none of the methods that a TDataset (1487) component has. It can be used to execute SQL statements on a database that update data, execute stored procedures and DDL statements etc.

The `TCustomSQLStatement` is equivalent to TSQLQuery (1529) in that it supports transactions (in the Transaction (1521) property) and parameters (in the Params (1525) property) and as such is a more versatile tool than executing queries using TSQLConnection.ExecuteDirect (1516).

To use a TCustomSQLStatement is simple and similar to the use of TSQLQuery (1529): set the Database (1555) property to an existing connection component, and set the Transaction (1557) property. After setting the SQL (1557) property and filling Params (1556), the SQL statement can be executed with the Execute (1512) method.

TCustomSQLStatement is a parent class. Many of the properties are only made public (or published) in the TSQLStatement (1554) class, which should be instantiated instead of the TCustomSQLStatement class.

See also: TSQLStatement (1554), TDataset (1487), TSQLQuery (1529), TSQLStatement.Transaction (1557), TSQLStatement.Params (1556), TCustomSQLStatement.Execute (1512), TSQLStatement.Database (1555), TSQLConnection.ExecuteDirect (1516)

60.14.2 Method overview

Page	Method	Description
1511	Create	Create a new instance of TCustomSQLStatement.
1511	Destroy	Destroy a TCustomSQLStatement instance.
1512	Execute	Execute the SQL statement.
1513	ParamByName	Find a parameter by name.
1512	Prepare	Prepare the statement for execution.
1513	RowsAffected	Number of rows affected by the SQL statement.
1512	Unprepare	Unprepare a previously prepared statement.

60.14.3 Property overview

Page	Properties	Access	Description
1513	Prepared	r	Is the statement prepared or not.

60.14.4 TCustomSQLStatement.Create

Synopsis: Create a new instance of TCustomSQLStatement.

Declaration:
```
constructor Create(AOwner: TComponent);  Override
```

Visibility: public

Description: Create initializes a new instance of TCustomSQLStatement and sets the SQL (1557)Params (1556), ParamCheck (1555) and ParseSQL (1557) to their initial values.

See also: TSQLStatement.SQL (1557), TSQLStatement.Params (1556), TSQLStatement.ParamCheck (1555), TSQLStatement.ParseSQL (1557), TSQLStatement.Destroy (1554)

60.14.5 TCustomSQLStatement.Destroy

Synopsis: Destroy a TCustomSQLStatement instance.

Declaration:
```
destructor Destroy;  Override
```

Visibility: public

Description: Destroy disconnects the TCustomSQLStatement instance from the transaction and database, and then frees the memory taken by the instance and its properties.

See also: TSQLStatement.Database (1555), TSQLStatement.Transaction (1557)

60.14.6 TCustomSQLStatement.Prepare

Synopsis: Prepare the statement for execution.

Declaration: `procedure Prepare`

Visibility: public

Description: `Prepare` prepares the SQL statement for execution. It is called automatically if Execute (1512) is called and the statement was not yet prepared. Depending on the database engine, it will also allocate the necessary resources on the database server.

Errors: An exception is raised if there is no SQL (1557) statement set or the Database (1555) or Transaction (1557) properties are empty.

See also: TSQLStatement.SQL (1557), TSQLStatement.Database (1555), TSQLStatement.Transaction (1557), TCustomSQLStatement.Execute (1512)

60.14.7 TCustomSQLStatement.Execute

Synopsis: Execute the SQL statement.

Declaration: `procedure Execute`

Visibility: public

Description: `Execute` executes the SQL (1557) statement on the database. If necessary, it will first open the connection and start a transaction, followed by a call to `Prepare`.

Errors: An exception is raised if there is no SQL (1557) statement set or the Database (1555) or Transaction (1557) properties are empty.

If an error occurs at the database level (the SQL failed to execute properly) then an exception is raised as well.

See also: TSQLStatement.SQL (1557), TSQLStatement.Database (1555), TSQLStatement.Transaction (1557)

60.14.8 TCustomSQLStatement.Unprepare

Synopsis: Unprepare a previously prepared statement.

Declaration: `procedure Unprepare`

Visibility: public

Description: `Unprepare` unprepares a prepared SQL statement. It is called automatically when the SQL statement is changed. Depending on the database engine, it will also de-allocate any allocated resources on the database server. if the statement is not in a prepared state, nothing happens.

Errors: If an error occurs at the database level (the unprepare operation failed to execute properly) then an exception is raised.

See also: TSQLStatement.SQL (1557), TSQLStatement.Database (1555), TSQLStatement.Transaction (1557), TCustomSQLStatement.Prepare (1512)

60.14.9 TCustomSQLStatement.ParamByName

Synopsis: Find a parameter by name.

Declaration: `function ParamByName(const AParamName: string) : TParam`

Visibility: public

Description: `ParamByName` finds the parameter `AParamName` in the Params (1556) property.

Errors: If no parameter with the given name is found, an exception is raised.

See also: TSQLStatement.Params (1556), TParams.ParamByname (1487)

60.14.10 TCustomSQLStatement.RowsAffected

Synopsis: Number of rows affected by the SQL statement.

Declaration: `function RowsAffected : TRowsCount;  Virtual`

Visibility: public

Description: `RowsAffected` is set to the number of affected rows after Execute (1512) was called. Not all databases may support this.

See also: TCustomSQLStatement.Execute (1512)

60.14.11 TCustomSQLStatement.Prepared

Synopsis: Is the statement prepared or not.

Declaration: `Property Prepared : Boolean`

Visibility: public

Access: Read

Description: `Prepared` equals `True` if Prepare (1512) was called (implicitly or explicitly), it returns `False` if not. It can be set to `True` or `False` to call Prepare (1512) or UnPrepare (1512), respectively.

See also: TCustomSQLStatement.Prepare (1512), TCustomSQLStatement.UnPrepare (1512)

60.15 TServerIndexDefs

60.15.1 Description

`TServerIndexDefs` is a simple descendent of TIndexDefs (599) that implements the necessary methods to update the list of definitions using the TSQLConnection (1514). It should not be used directly.

See also: TSQLConnection (1514)

60.15.2 Method overview

Page	Method	Description
1514	Create	Create a new instance of `TServerIndexDefs`.
1514	Update	Updates the list of indexes.

60.15.3 TServerIndexDefs.Create

Synopsis: Create a new instance of `TServerIndexDefs`.

Declaration: `constructor Create(ADataSet: TDataSet); Override`

Visibility: public

Description: `Create` will raise an exception if `ADataset` is not a TCustomSQLQuery (1505) descendent.

Errors: An `EDatabaseError` exception will be raised if `ADataset` is not a TCustomSQLQuery (1505) descendent.

60.15.4 TServerIndexDefs.Update

Synopsis: Updates the list of indexes.

Declaration: `procedure Update; Override`

Visibility: public

Description: `Update` updates the list of indexes, it uses the TSQLConnection (1514) methods for this.

60.16 TSQLConnection

60.16.1 Description

`TSQLConnection` is an abstract class for making a connection to a SQL Database. This class will never be instantiated directly, for each database type a descendent class specific for this database type must be created.

Most of common properties to SQL databases are implemented in this class.

See also: TSQLQuery (1529), TSQLTransaction (1558)

60.16.2 Method overview

Page	Method	Description
1515	Create	Create a new instance of `TSQLConnection`.
1519	CreateDB	Create a new Database on the server.
1515	Destroy	Destroys the instance of the connection.
1519	DropDB	Procedure to drop or remove a Database.
1516	EndTransaction	End the Transaction associated with this connection.
1516	ExecuteDirect	Execute a piece of SQL code directly, using a Transaction if specified.
1518	GetConnectionInfo	Return some information about the connection.
1518	GetFieldNames	Gets a list of the field names in the specified table.
1519	GetNextValue	Get next value for a sequence.
1517	GetObjectNames	Return a collection of object names for a given type of object.
1517	GetProcedureNames	Gets a list of Stored Procedures in the Database.
1518	GetSchemaNames	Get database schema names.
1518	GetSequenceNames	Return a list of sequence names.
1519	GetStatementInfo	Get statement information.
1517	GetTableNames	Get a list of the tables in the specified database.
1517	HasTable	
1516	StartTransaction	Start the Transaction associated with this Connection.

60.16.3 Property overview

Page	Properties	Access	Description
1522	CharSet	rw	The character set to be used in this database.
1524	Connected		Is a connection to the server active or not.
1520	ConnOptions	r	The set of Connection options being used in the Connection.
1524	DatabaseName		The name of the database to which connection is required.
1520	FieldNameQuoteChars	rw	Characters used to quote field names.
1520	Handle	r	Low level handle used by the connection.
1522	HostName	rw	The name of the host computer where the database resides.
1524	KeepConnection		Attempt to keep the connection open once it is established.
1523	LogEvents	rw	Filter for events to log.
1525	LoginPrompt		Should SQLDB prompt for user credentials when a connection is activated.
1522	OnLog	rw	Event handler for logging events.
1525	OnLogin		Event handler for login process.
1523	Options	rw	Options to observe for this connection.
1525	Params		Extra connection parameters.
1521	Password	rw	Password used when authenticating on the database server.
1523	Role	rw	Role in which the user is connecting to the database.
1521	Transaction	rw	Default transaction to be used for this connection.
1521	UserName	rw	The username for authentication on the database server.

60.16.4 TSQLConnection.Create

Synopsis: Create a new instance of `TSQLConnection`.

Declaration: `constructor Create(AOwner: TComponent); Override`

Visibility: public

Description: `Create` initialized a new instance of TSQLconnection (1514). After calling the inherited constructor, it will initialize the FieldNameQuoteChars (1520) property and some other fields for internal use.

See also: FieldNameQuoteChars (1520)

60.16.5 TSQLConnection.Destroy

Synopsis: Destroys the instance of the connection.

Declaration: `destructor Destroy; Override`

Visibility: public

Description: `Destroy` removes the connection from memory. When a connection is removed, all datasets are closed, and all transactions too.

60.16.6 TSQLConnection.StartTransaction

Synopsis: Start the Transaction associated with this Connection.

Declaration:
```
procedure StartTransaction;  Override
```

Visibility: public

Description: `StartTransaction` is a convenience method which starts the default transaction (Transaction (1521)). It is equivalent to

```
Connection.Transaction.StartTransaction
```

Errors: If no transaction is assigned, an exception will be raised.

See also: EndTransaction (1516)

60.16.7 TSQLConnection.EndTransaction

Synopsis: End the Transaction associated with this connection.

Declaration:
```
procedure EndTransaction;  Override
```

Visibility: public

Description: `StartTransaction` is a convenience method which ends the default transaction (TSQLConnection.Transaction (1521)). It is equivalent to

```
Connection.Transaction.EndTransaction
```

Errors: If no transaction is assigned, an exception will be raised.

See also: StartTransaction (1516)

60.16.8 TSQLConnection.ExecuteDirect

Synopsis: Execute a piece of SQL code directly, using a Transaction if specified.

Declaration:
```
procedure ExecuteDirect(SQL: string);  Virtual;  Overload
procedure ExecuteDirect(SQL: string; ATransaction: TSQLTransaction)
                        ;  Virtual;  Overload
```

Visibility: public

Description: `ExecuteDirect` executes an SQL statement directly. If `ATransaction` is `Nil` then the default transaction is used, otherwise the specified transaction is used.

`ExecuteDirect` does not offer support for parameters, so only statements that do not need parsing and parameters substitution can be handled. If parameter substitution is required, use a TSQLQuery (1529) component and its ExecSQL (1507) method.

Errors: If no transaction is assigned, and no transaction is passed, an exception will be raised.

See also: TSQLQuery (1529), ExecSQL (1507)

60.16.9 TSQLConnection.GetObjectNames

Synopsis: Return a collection of object names for a given type of object.

Declaration:
```
function GetObjectNames(ASchemaType: TSchemaType;
                        AList: TSqlObjectIdentifierList) : Integer
                        ;  Virtual
```

Visibility: public

Description: `GetObjectNames` returns all the names of objects of a given type `ASchemaType` in the database, and returns the number of found objects as a result. The object names are placed in the collection `AList`.

The collection is not cleared.

See also: #fcl.sqltypes.TSqlObjectIdentifierList (1566)

60.16.10 TSQLConnection.HasTable

Declaration:
```
function HasTable(const aTable: string; SearchSystemTables: Boolean)
                  : Boolean
```

Visibility: public

60.16.11 TSQLConnection.GetTableNames

Synopsis: Get a list of the tables in the specified database.

Declaration:
```
procedure GetTableNames(List: TStrings; SystemTables: Boolean);  Virtual
```

Visibility: public

Description: `GetTableNames` will return the names of the tables in the database in `List`. If `SystemTables` is `True` then only the names of system tables will be returned.

`List` is cleared before adding the names.

Remark Note that the list may depend on the access rights of the user.

See also: TSQLConnection.GetProcedureNames (1517), TSQLConnection.GetFieldNames (1518)

60.16.12 TSQLConnection.GetProcedureNames

Synopsis: Gets a list of Stored Procedures in the Database.

Declaration:
```
procedure GetProcedureNames(List: TStrings);  Virtual
```

Visibility: public

Description: `GetProcedureNames` will return the names of the stored procedures in the database in `List`.

`List` is cleared before adding the names.

See also: TSQLConnection.GetTableNames (1517), TSQLConnection.GetFieldNames (1518)

60.16.13 TSQLConnection.GetFieldNames

Synopsis: Gets a list of the field names in the specified table.

Declaration:
```
procedure GetFieldNames(const TableName: string; List: TStrings)
                        ;  Virtual
```

Visibility: public

Description: `GetFieldNames` will return the names of the fields in `TableName` in `list` `List` is cleared before adding the names.

Errors: If a non-existing tablename is passed, no error will be raised.

See also: TSQLConnection.GetTableNames (1517), TSQLConnection.GetProcedureNames (1517)

60.16.14 TSQLConnection.GetSchemaNames

Synopsis: Get database schema names.

Declaration:
```
procedure GetSchemaNames(List: TStrings);  Virtual
```

Visibility: public

Description: `GetSchemaNames` returns a list of schemas defined in the database.

See also: TSQLConnection.GetTableNames (1517), TSQLConnection.GetProcedureNames (1517), TSQLConnection.GetFieldNames (1518)

60.16.15 TSQLConnection.GetSequenceNames

Synopsis: Return a list of sequence names.

Declaration:
```
procedure GetSequenceNames(List: TStrings);  Virtual
```

Visibility: public

Description: `GetSequenceNames` returns the names of all defined sequences (Generators in Firebird) in the databases in `List`, if the database engine supports them: Not all database types support sequences.

This call is a convenience call, a simple wrapper used to call the `GetDBInfo` method.

Errors: None.

60.16.16 TSQLConnection.GetConnectionInfo

Synopsis: Return some information about the connection.

Declaration:
```
function GetConnectionInfo(InfoType: TConnInfoType) : string;  Virtual
```

Visibility: public

Description: `GetConnectionInfo` can be used to return some information about the connection. Which information is returned depends on the `InfoType` parameter. The information is returned as a string. If `citAll` is passed, then the result will be a comma-separated list of values, each of the values enclosed in double quotes.

See also: TConnInfoType (1497)

60.16.17 TSQLConnection.GetStatementInfo

Synopsis: Get statement information.

Declaration:
```
function GetStatementInfo(const ASQL: string) : TSQLStatementInfo
                          ; Virtual
```

Visibility: public

Description: `GetStatementInfo` returns information about the `ASQL` SQL command. To this end, it will partially parse the statement.

See also: TSQLStatementInfo (1502)

60.16.18 TSQLConnection.CreateDB

Synopsis: Create a new Database on the server.

Declaration:
```
procedure CreateDB; Virtual
```

Visibility: public

Description: `CreateDB` will create a new database on the server. Whether or not this functionality is present depends on the type of the connection. The name for the new database is taken from the TSQLConnection.DatabaseName (1524) property, the user credentials are taken from the TSQLConnection.UserName (1521) and TSQLConnection.Password (1521) properties.

Errors: If the connection type does not support creating a database, then an `EDatabaseError` exception is raised. Other exceptions may be raised if the operation fails, e.g. when the user does not have the necessary access rights.

See also: TSQLConnection.DropDB (1519)

60.16.19 TSQLConnection.DropDB

Synopsis: Procedure to drop or remove a Database.

Declaration:
```
procedure DropDB; Virtual
```

Visibility: public

Description: `DropDB` does the opposite of CreateDB (1519). It removes the database from the server. The database must be connected before this command may be used. Whether or not this functionality is present depends on the type of the connection.

Errors: If the connection type does not support creating a database, then an `EDatabaseError` exception is raised. Other exceptions may be raised if the operation fails, e.g. when the user does not have the necessary access rights.

See also: TSQLConnection.CreateDB (1519)

60.16.20 TSQLConnection.GetNextValue

Synopsis: Get next value for a sequence.

Declaration:
```
function GetNextValue(const SequenceName: string; IncrementBy: Integer)
                      : Int64; Virtual
```

Visibility: public

Description: `GetNextValue` returns the next value for the sequence `SequenceName`, incrementing the current value with `IncrementBy` (default 1).

Errors: Not all databases support sequences, in that case an SQL error will be raised.

See also: TSQLConnection.GetSequenceNames (1518)

60.16.21 TSQLConnection.ConnOptions

Synopsis: The set of Connection options being used in the Connection.

Declaration: `Property ConnOptions : TConnOptions`

Visibility: public

Access: Read

Description: `ConnOptions` is the set of options used by this connection component. It is normally the same value for all connections of the same type

See also: TConnOption (1497)

60.16.22 TSQLConnection.Handle

Synopsis: Low level handle used by the connection.

Declaration: `Property Handle : Pointer`

Visibility: public

Access: Read

Description: `Handle` represents the low-level handle that the TSQLCOnnection component has received from the client library of the database. Under normal circumstances, this property must not be used.

60.16.23 TSQLConnection.FieldNameQuoteChars

Synopsis: Characters used to quote field names.

Declaration: `Property FieldNameQuoteChars : TQuoteChars`

Visibility: public

Access: Read,Write

Description: `FieldNameQuoteChars` can be set to specify the characters that should be used to delimit field names in SQL statements generated by SQLDB. It is normally initialized correctly by the TSQLConnection (1514) descendent to the default for that particular connection type.

See also: TSQLConnection (1514)

60.16.24 TSQLConnection.Password

Synopsis: Password used when authenticating on the database server.

Declaration: `Property Password : string`

Visibility: published

Access: Read,Write

Description: `Password` is used when authenticating the user specified in UserName (1521) when connecting to the database server

This property must be set prior to activating the connection. Changing it while the connection is active has no effect.

See also: TSQLConnection.UserName (1521), TSQLConnection.HostName (1522)

60.16.25 TSQLConnection.Transaction

Synopsis: Default transaction to be used for this connection.

Declaration: `Property Transaction : TSQLTransaction`

Visibility: published

Access: Read,Write

Description: `Transaction` should be set to a TSQLTransaction (1558) instance. It is set as the default transaction when a query is connected to the database, and is used in several metadata operations such as TSQLConnection.GetTableNames (1517)

See also: TSQLTransaction (1558)

60.16.26 TSQLConnection.UserName

Synopsis: The username for authentication on the database server.

Declaration: `Property UserName : string`

Visibility: published

Access: Read,Write

Description: `UserName` is used to authenticate on the database server when the connection to the database is established.

This property must be set prior to activating the connection. Changing it while the connection is active has no effect.

See also: TSQLConnection.Password (1521), TSQLConnection.HostName (1522), TSQLConnection.Role (1523), TSQLConnection.Charset (1522)

60.16.27 TSQLConnection.CharSet

Synopsis: The character set to be used in this database.

Declaration: `Property CharSet : string`

Visibility: published

Access: Read,Write

Description: `Charset` can be used to tell the user in which character set the data will be sent to the server, and in which character set the results should be sent to the client. Some connection types will ignore this property, and the data will be sent to the client in the encoding used on the server.

This property must be set prior to activating the connection. Changing it while the connection is active has no effect.

Remark SQLDB will not do anything with this setting except pass it on to the server if a specific connection type supports it. It does not perform any conversions by itself based on the value of this setting.

See also: TSQLConnection.Password (1521), TSQLConnection.HostName (1522), TSQLConnection.UserName (1521), TSQLConnection.Role (1523)

60.16.28 TSQLConnection.HostName

Synopsis: The name of the host computer where the database resides.

Declaration: `Property HostName : string`

Visibility: published

Access: Read,Write

Description: `HostName` is the name of the host computer where the database server is listening for connection. An empty value means the local machine is used.

This property must be set prior to activating the connection. Changing it while the connection is active has no effect.

See also: TSQLConnection.Role (1523), TSQLConnection.Password (1521), TSQLConnection.UserName (1521), TSQLConnection.DatabaseName (1524), TSQLConnection.Charset (1522)

60.16.29 TSQLConnection.OnLog

Synopsis: Event handler for logging events.

Declaration: `Property OnLog : TDBLogNotifyEvent`

Visibility: published

Access: Read,Write

Description: `TSQLConnection` can send events for all the actions that it performs: executing SQL statements, commit and rollback of transactions etc. This event handler must be set to react on these events: they can for example be written to a log file. Only events specified in the LogEvents (1523) property will be logged.

The events received by this event handler are specific for this connection. To receive events from all active connections in the application, set the global GlobalDBLogHook (1500) event handler.

See also: GlobalDBLogHook (1500), TSQLConnection.LogEvents (1523)

60.16.30 TSQLConnection.LogEvents

Synopsis: Filter for events to log.

Declaration: `Property LogEvents : TDBEventTypes`

Visibility: published

Access: Read,Write

Description: `LogEvents` can be used to filter the events which should be sent to the OnLog (1522) and GlobalDBLogHook (1500) event handlers. Only event types that are listed in this property will be sent.

See also: GlobalDBLogHook (1500), TSQLConnection.OnLog (1522)

60.16.31 TSQLConnection.Options

Synopsis: Options to observe for this connection.

Declaration: `Property Options : TSQLConnectionOptions`

Visibility: published

Access: Read,Write

Description: `Options` can be used to control the behaviour of SQLDB for this connection. The following options can be set:

scoExplicitConnect When set, the connection must be explicitly made. Default behaviour is for `TSQLQuery` to implicitly open the connection as needed.

scoApplyUpdatesChecksRowsAffected When set, whenever an update SQL Statement is executed during ApplyOptions of a dataset, the RowsAffected (1513) is checked and must be equal to 1.

See also: TCustomSQLQuery.ApplyUpdates (1509), TCustomSQLStatement.RowsAffected (1513), TCustomSQLQuery.RowsAffected (1508)

60.16.32 TSQLConnection.Role

Synopsis: Role in which the user is connecting to the database.

Declaration: `Property Role : string`

Visibility: published

Access: Read,Write

Description: `Role` is used to specify the user's role when connecting to the database user. Not all connection types support roles, for those that do not, this property is ignored.

This property must be set prior to activating the connection. Changing it while the connection is active has no effect.

See also: TSQLConnection.Password (1521), TSQLConnection.UserName (1521), TSQLConnection.DatabaseName (1524), TSQLConnection.Hostname (1522)

60.16.33 TSQLConnection.Connected

Synopsis: Is a connection to the server active or not.

Declaration: `Property Connected :`

Visibility: published

Access:

Description: `Connected` indicates whether a connection to the server is active or not. No queries to this server can be activated as long as the value is `False`

Setting the property to `True` will attempt a connection to the database DatabaseName (1524) on host HostName (1522) using the credentials specified in UserName (1521) and Password (1521). If the connection or authentication fails, an exception is raised. This has the same effect as calling Open (483).

Setting the property to `False` will close the connection to the database. All datasets connected to the database will be closed, all transactions will be closed as well. This has the same effect as calling Close (1487)

See also: TSQLConnection.Password (1521), TSQLConnection.UserName (1521), TSQLConnection.DatabaseName (1524), TSQLConnection.Role (1523)

60.16.34 TSQLConnection.DatabaseName

Synopsis: The name of the database to which connection is required.

Declaration: `Property DatabaseName :`

Visibility: published

Access:

Description: `DatabaseName` is the name of the database to which a connection must be made. Some servers need a complete path to a file, others need a symbolic name (an alias): the interpretation of this name depends on the connection type.

This property must be set prior to activating the connection. Changing it while the connection is active has no effect.

See also: TSQLConnection.Password (1521), TSQLConnection.UserName (1521), TSQLConnection.Charset (1522), TSQLConnection.Hostname (1522)

60.16.35 TSQLConnection.KeepConnection

Synopsis: Attempt to keep the connection open once it is established.

Declaration: `Property KeepConnection :`

Visibility: published

Access:

Description: `KeepConnection` can be used to attempt to keep the connection open once it is established. This property is currently not implemented.

60.16.36 TSQLConnection.LoginPrompt

Synopsis: Should SQLDB prompt for user credentials when a connection is activated.

Declaration: `Property LoginPrompt :`

Visibility: published

Access:

Description: `LoginPrompt` can be set to `True` to force the system to get a username/password pair from the user. How these data are fetched from the used depends on the OnLogin (1525) event handler. The UserName (1521) and Password (1521) properties are ignored in this case.

See also: TSQLConnection.Password (1521), TSQLConnection.UserName (1521), OnLogin (1525)

60.16.37 TSQLConnection.Params

Synopsis: Extra connection parameters.

Declaration: `Property Params :`

Visibility: published

Access:

Description: `Params` can be used to specify extra parameters to use when establishing a connection to the database. Which parameters can be specified depends on the connection type.

See also: TSQLConnection.Password (1521), TSQLConnection.UserName (1521), TSQLConnection.Hostname (1522), TSQLConnection.DatabaseName (1524)

60.16.38 TSQLConnection.OnLogin

Synopsis: Event handler for login process.

Declaration: `Property OnLogin :`

Visibility: published

Access:

Description: `OnLogin` will be used when loginPrompt (1525) is `True`. It will be called, and can be used to present a user with a dialog in which the username and password can be asked.

See also: TSQLConnection.LoginPrompt (1525)

60.17 TSQLConnector

60.17.1 Description

`TSQLConnector` implements a general connection type. When switching database backends, the normal procedure is to replace one instance of TSQLConnection (1514) descendent with another, and connect all instances of TSQLQuery (1529) and TSQLTransaction (1558) to the new connection.

Using `TSQLConnector` avoids this: the type of connection can be set using the ConnectorType (1526) property, which is a string property. The `TSQLConnector` class will (in the background) create the correct TSQLConnection (1514) descendent to handle all actual operations on the database.

In all other respects, TSQLConnector acts like a regular TSQLConnection instance. Since no access to the actually used TSQLConnection descendent is available, connection-specific calls are not available.

See also: TSQLConnector.ConnectorType (1526), UniversalConnectors (1490)

60.17.2 Property overview

Page	Properties	Access	Description
1526	ConnectorType	rw	Name of the connection type to use.
1526	Port		

60.17.3 TSQLConnector.ConnectorType

Synopsis: Name of the connection type to use.

Declaration: `Property ConnectorType : string`

Visibility: published

Access: Read,Write

Description: ConnectorType should be set to one of the available connector types in the application. The list of possible connector types can be retrieved using GetConnectionList (1501) call. The ConnectorType property can only be set when the connection is not active.

Errors: Attempting to change the ConnectorType property while the connection is active will result in an exception.

See also: GetConnectionList (1501)

60.17.4 TSQLConnector.Port

Declaration: `Property Port :`

Visibility: published

Access:

60.18 TSQLCursor

60.18.1 Description

TSQLCursor is an abstract internal object representing a result set returned by a single SQL select statement (TSQLHandle (1528)). statement. It is used by the TSQLQuery (1529) component to handle result sets returned by SQL statements.

This object must not be used directly.

See also: TSQLQuery (1529), TSQLHandle (1528)

60.19 TSQLDBFieldDef

60.19.1 Description

`TSQLDBFieldDef` is a SQLDB specific db.TFieldDef (1487) descendent which has room for storing engine-specific data for the result set fields.

See also: TSQLDBFieldDef.SQLDBData (1527)

60.19.2 Property overview

Page	Properties	Access	Description
1527	SQLDBData	rw	Pointer to store engine-specific data for the result field.

60.19.3 TSQLDBFieldDef.SQLDBData

Synopsis: Pointer to store engine-specific data for the result field.

Declaration: `Property SQLDBData : Pointer`

Visibility: public

Access: Read,Write

Description: `SQLDBData` can be used by the TSQLConnection (1514) descendents to store additional data about fields in a result set. It is not used by `TSQLQuery` itself.

See also: TSQLQuery (1529)

60.20 TSQLDBFieldDefs

60.20.1 Description

`TSQLDBFieldDefs` is a `TFieldDefs` descendent which creates TSQLDBFieldDef (1527) descendents when a new field is added to the field set, to provide storage for engine-specific field data.

See also: TSQLDBFieldDef (1527)

60.21 TSQLDBParam

60.21.1 Description

`TSQLDBParam` is used to be able to create parameters which can store info about a field on which the parameter is based in the TSQLDBParam.FieldDef (1528) field. This is useful when constructing update or insert queries. It can also store private data needed for the various TSQLConnection (1514) descendents in the TSQLDBParam.SQLDBData (1528) property.

See also: TSQLDBFieldDef (1527), TSQLDBParam.FieldDef (1528), TSQLDBParam.SQLDBData (1528)

60.21.2 Property overview

Page	Properties	Access	Description
1528	FieldDef	rw	Field definition for update SQL.
1528	SQLDBData	rw	Private data for TSQLDB descendents.

60.21.3 TSQLDBParam.FieldDef

Synopsis: Field definition for update SQL.

Declaration: `Property FieldDef : TFieldDef`

Visibility: public

Access: Read,Write

Description: `FieldDef` is used during generation of update SQL statements to store information about the field to be updated.

See also: TSQLDBFieldDef (1527), TSQLDBParam.SQLDBData (1528)

60.21.4 TSQLDBParam.SQLDBData

Synopsis: Private data for TSQLDB descendents.

Declaration: `Property SQLDBData : Pointer`

Visibility: public

Access: Read,Write

Description: `SQLDBData` should not be used by the end-user, it is for internal use by the TSQLConnection (1514) descendents.

See also: TSQLConnection (1514), TSQLDBParam.FieldDef (1528)

60.22 TSQLDBParams

60.22.1 Description

`TSQLDBParams` is a `TParams` descendent which creates TSQLDBParams (1528) descendents when a new field is added to the field set, to provide storage for engine-specific field data.

See also: TSQLDBParam (1527)

60.23 TSQLHandle

60.23.1 Description

`TSQLHandle` is an abstract internal object representing a database client handle. It is used by the various connections to implement the connection-specific functionality, and usually represents a low-level handle. It is used by the TSQLQuery (1529) component to communicate with the TSQL-Connection (1514) descendent.

This object must not be used directly.

See also: TSQLQuery (1529), TSQLCursor (1526)

60.24 TSQLQuery

60.24.1 Description

`TSQLQuery` exposes the properties and some methods introduced in TCustomSQLQuery (1505). It encapsulates a single SQL statement: it implements all the necessary #fcl.db.TDataset (496) functionality to be able to handle a result set. It can also be used to execute a single SQL statement that does not return data, using the TCustomSQLQuery.ExecSQL (1507) method.

Typically, the TSQLQuery.Database (1537) property must be set once, the TSQLQuery.Transaction (1537) property as well. Then the TSQLQuery.SQL (1537) property can be set. Depending on the kind of SQL statement, the Open (515) method can be used to retrieve data, or the `ExecSQL` method can be used to execute the SQL statement (this can be used for DDL statements, or update statements).

See also: TSQLTransaction (1558), TSQLConnection (1514), TCustomSQLQuery.ExecSQL (1507), TSQLQuery.SQL (1537)

60.24.2 Property overview

Page	Properties	Access	Description
1533	Active		
1533	AfterCancel		
1533	AfterClose		
1533	AfterDelete		
1534	AfterEdit		
1534	AfterInsert		
1534	AfterOpen		
1534	AfterPost		
1534	AfterRefresh		Event triggered after refresh.
1534	AfterScroll		
1533	AutoCalcFields		
1534	BeforeCancel		
1535	BeforeClose		
1535	BeforeDelete		
1535	BeforeEdit		
1535	BeforeInsert		
1535	BeforeOpen		
1535	BeforePost		
1535	BeforeRefresh		Event triggered before refresh.
1536	BeforeScroll		
1537	Database		The `TSQLConnection` instance on which to execute SQL Statements.
1543	DataSource		Source for parameter values for unbound parameters.
1539	DeleteSQL		Statement to be used when deleting a new row in the database.
1532	FieldDefs		List of field definitions.
1533	Filter		
1533	Filtered		
1539	IndexDefs		List of local index Definitions.
1538	InsertSQL		Statement to be used when inserting a new row in the database.
1541	MacroChar		Macro delimiter character.
1541	MacroCheck		Check for macros in the SQL statement.
1541	Macros		Set of macros for this SQL statement.
1532	MaxIndexesCount		Maximum allowed number of indexes.
1536	OnCalcFields		
1536	OnDeleteError		
1536	OnEditError		
1536	OnFilterRecord		
1536	OnNewRecord		
1536	OnPostError		
1540	Options		Options controlling the behaviour of the dataset.
1541	ParamCheck		Should the SQL statement be checked for parameters.
1540	Params		Parameters detected in the SQL statement.
1542	ParseSQL		Should the SQL statement be parsed or not.
1537	ReadOnly		
1539	RefreshSQL		Refresh query to re-fetch field values after a DB update.
1532	SchemaType		Schema type.
1543	Sequence		Sequence to use for auto-generating values using a sequence.
1544	ServerFilter		Append server-side filter to SQL statement.
1544	ServerFiltered		Should server-side filter be applied.
1544	ServerIndexDefs		List of indexes on the primary table of the query.
1537	SQL		The SQL statement to execute.
1532	StatementType		SQL statement type.
1537	Transaction		Transaction in which to execute SQL statements.
1542	UpdateMode		How to create update SQL statements.
1538	UpdateSQL		Statement to be used when updating an existing row

60.24.3 TSQLQuery.SchemaType

Synopsis: Schema type.

Declaration: `Property SchemaType :`

Visibility: public

Access:

Description: `SchemaType` is the schema type set by TCustomSQLQuery.SetSchemaInfo (1508). It determines what kind of schema information will be returned by the `TSQLQuery` instance.

See also: TCustomSQLQuery.SetSchemaInfo (1508), RetrievingSchemaInformation (1491)

60.24.4 TSQLQuery.StatementType

Synopsis: SQL statement type.

Declaration: `Property StatementType :`

Visibility: public

Access:

Description: `StatementType` is determined during the Prepare (1507) call when ParseSQL (1542) is set to `True`. It gives an indication of the type of SQL statement that is being executed.

See also: TSQLQuery.SQL (1537), TSQLQuery.ParseSQL (1542), TSQLQuery.Params (1540)

60.24.5 TSQLQuery.MaxIndexesCount

Synopsis: Maximum allowed number of indexes.

Declaration: `Property MaxIndexesCount :`

Visibility: published

Access:

Description: `MaxIndexesCount` determines the number of index entries that the dataset will reserve for indexes. No more indexes than indicated here can be used. The property must be set before the dataset is opened. The minimum value for this property is 1. The default value is 2.

If an index is added and the current index count equals `MaxIndexesCount`, an exception will be raised.

Errors: Attempting to set this property while the dataset is active will raise an exception.

60.24.6 TSQLQuery.FieldDefs

Synopsis: List of field definitions.

Declaration: `Property FieldDefs :`

Visibility: published

Access:

60.24.7 TSQLQuery.Active

Declaration: `Property Active :`

Visibility: published

Access:

60.24.8 TSQLQuery.AutoCalcFields

Declaration: `Property AutoCalcFields :`

Visibility: published

Access:

60.24.9 TSQLQuery.Filter

Declaration: `Property Filter :`

Visibility: published

Access:

60.24.10 TSQLQuery.Filtered

Declaration: `Property Filtered :`

Visibility: published

Access:

60.24.11 TSQLQuery.AfterCancel

Declaration: `Property AfterCancel :`

Visibility: published

Access:

60.24.12 TSQLQuery.AfterClose

Declaration: `Property AfterClose :`

Visibility: published

Access:

60.24.13 TSQLQuery.AfterDelete

Declaration: `Property AfterDelete :`

Visibility: published

Access:

60.24.14 TSQLQuery.AfterEdit

Declaration: `Property AfterEdit :`

Visibility: published

Access:

60.24.15 TSQLQuery.AfterInsert

Declaration: `Property AfterInsert :`

Visibility: published

Access:

60.24.16 TSQLQuery.AfterOpen

Declaration: `Property AfterOpen :`

Visibility: published

Access:

60.24.17 TSQLQuery.AfterPost

Declaration: `Property AfterPost :`

Visibility: published

Access:

60.24.18 TSQLQuery.AfterRefresh

Synopsis: Event triggered after refresh.

Declaration: `Property AfterRefresh :`

Visibility: published

Access:

60.24.19 TSQLQuery.AfterScroll

Declaration: `Property AfterScroll :`

Visibility: published

Access:

60.24.20 TSQLQuery.BeforeCancel

Declaration: `Property BeforeCancel :`

Visibility: published

Access:

60.24.21 TSQLQuery.BeforeClose

Declaration: `Property BeforeClose :`

Visibility: published

Access:

60.24.22 TSQLQuery.BeforeDelete

Declaration: `Property BeforeDelete :`

Visibility: published

Access:

60.24.23 TSQLQuery.BeforeEdit

Declaration: `Property BeforeEdit :`

Visibility: published

Access:

60.24.24 TSQLQuery.BeforeInsert

Declaration: `Property BeforeInsert :`

Visibility: published

Access:

60.24.25 TSQLQuery.BeforeOpen

Declaration: `Property BeforeOpen :`

Visibility: published

Access:

60.24.26 TSQLQuery.BeforePost

Declaration: `Property BeforePost :`

Visibility: published

Access:

60.24.27 TSQLQuery.BeforeRefresh

Synopsis: Event triggered before refresh.

Declaration: `Property BeforeRefresh :`

Visibility: published

Access:

60.24.28 TSQLQuery.BeforeScroll

Declaration: `Property BeforeScroll :`

Visibility: published

Access:

60.24.29 TSQLQuery.OnCalcFields

Declaration: `Property OnCalcFields :`

Visibility: published

Access:

60.24.30 TSQLQuery.OnDeleteError

Declaration: `Property OnDeleteError :`

Visibility: published

Access:

60.24.31 TSQLQuery.OnEditError

Declaration: `Property OnEditError :`

Visibility: published

Access:

60.24.32 TSQLQuery.OnFilterRecord

Declaration: `Property OnFilterRecord :`

Visibility: published

Access:

60.24.33 TSQLQuery.OnNewRecord

Declaration: `Property OnNewRecord :`

Visibility: published

Access:

60.24.34 TSQLQuery.OnPostError

Declaration: `Property OnPostError :`

Visibility: published

Access:

60.24.35 TSQLQuery.Database

Synopsis: The `TSQLConnection` instance on which to execute SQL Statements.

Declaration: `Property Database :`

Visibility: published

Access:

Description: `Database` is the SQL connection (of type TSQLConnection (1514)) on which SQL statements will be executed, and from which result sets will be retrieved. This property must be set before any form of SQL command can be executed, just like the Transaction (1537) property must be set.

Multiple `TSQLQuery` instances can be connected to a database at the same time.

See also: TSQLQuery.Transaction (1537), TSQLConnection (1514), TSQLTransaction (1558)

60.24.36 TSQLQuery.Transaction

Synopsis: Transaction in which to execute SQL statements.

Declaration: `Property Transaction :`

Visibility: published

Access:

Description: `Transaction` must be set to a SQL transaction (of type TSQLTransaction (1558)) component. All SQL statements (`SQL` / `InsertSQL` / `updateSQL` / `DeleteSQL`) etc.) will be executed in the context of this transaction.

The transaction must be connected to the same database instance as the query itself.

Multiple `TSQLQuery` instances can be connected to a transaction at the same time. If the transaction is rolled back, all changes done by all `TSQLQuery` instances will be rolled back.

See also: TSQLQuery.Database (1537), TSQLConnection (1514), TSQLTransaction (1558)

60.24.37 TSQLQuery.ReadOnly

Declaration: `Property ReadOnly :`

Visibility: published

Access:

60.24.38 TSQLQuery.SQL

Synopsis: The SQL statement to execute.

Declaration: `Property SQL :`

Visibility: published

Access:

Description: SQL is the SQL statement that will be executed when ExecSQL (1507) is called, or Open (515) is called. It should contain a valid SQL statement for the connection to which the TSQLQuery (1529) component is connected. SQLDB will not attempt to modify the SQL statement so it is accepted by the SQL engine.

Setting or modifying the SQL statement will call UnPrepare (1507)

If ParseSQL (1542) is True, the SQL statement will be parsed and the Params (1540) property will be updated with the names of the parameters found in the SQL statement.

See also Using parameters.

See also: TSQLQuery.ParseSQL (1542), TSQLQuery.Params (1540), TCustomSQLQuery.ExecSQL (1507), TDataset.Open (515)

60.24.39 TSQLQuery.InsertSQL

Synopsis: Statement to be used when inserting a new row in the database.

Declaration: Property InsertSQL :

Visibility: published

Access:

Description: InsertSQL can be used to specify an SQL INSERT statement, which is used when a new record was appended to the dataset, and the changes must be written to the database. TSQLQuery can generate an insert statement by itself for many cases, but in case it fails, the statement to be used for the insert can be specified here.

The SQL statement should be parameterized according to the conventions for specifying parameters. Note that old field values can be specified as :OLD_FIELDNAME

See also: TSQLQuery.SQL (1537), TSQLQuery.UpdateSQL (1538), TSQLQuery.DeleteSQL (1539), TSQLQuery.UpdateMode (1542), UsingParams (1492), UpdateSQLS (1491)

60.24.40 TSQLQuery.UpdateSQL

Synopsis: Statement to be used when updating an existing row in the database.

Declaration: Property UpdateSQL :

Visibility: published

Access:

Description: UpdateSQL can be used to specify an SQL UPDATE statement, which is used when an existing record was modified in the dataset, and the changes must be written to the database. TSQLQuery can generate an update statement by itself for many cases, but in case it fails, the statement to be used for the update can be specified here.

The SQL statement should be parameterized according to the conventions for specifying parameters. Note that old field values can be specified as :OLD_FIELDNAME

See also: TSQLQuery.SQL (1537), TSQLQuery.InsertSQL (1538), TSQLQuery.DeleteSQL (1539), TSQLQuery.UpdateMode (1542), UsingParams (1492), UpdateSQLS (1491)

60.24.41 TSQLQuery.DeleteSQL

Synopsis: Statement to be used when deleting a new row in the database.

Declaration: `Property DeleteSQL :`

Visibility: published

Access:

Description: `DeleteSQL` can be used to specify an SQL `DELETE` statement, which is used when an existing record was deleted from the dataset, and the changes must be written to the database. `TSQLQuery` can generate a delete statement by itself for many cases, but in case it fails, the statement to be used for the delete operation can be specified here.

The SQL statement should be parameterized according to the conventions for specifying parameters. Note that old field values can be specified as `:OLD_FIELDNAME`

See also: TSQLQuery.SQL (1537), TSQLQuery.UpdateSQL (1538), TSQLQuery.DeleteSQL (1539), TSQLQuery.UpdateMode (1542), UsingParams (1492), UpdateSQLS (1491)

60.24.42 TSQLQuery.RefreshSQL

Synopsis: Refresh query to re-fetch field values after a DB update.

Declaration: `Property RefreshSQL :`

Visibility: published

Access:

Description: `RefreshSQL` can be used to specify a SQL statement that is executed after an `UPDATE` or `INSERT` operation. The query will be executed, and the values of all fields in the result set will be copied to the dataset. This SQL statement is only executed during the `ApplyUpdates` operation, not during the `Post` call itself.

A `RefreshSQL` can be constructed automatically by SQLDB by setting the `pfRefreshOnUpdate` or `pfRefreshOnInsert` flags in the ProviderFlags (1487) of the fields in the dataset, depending on whether the operation was an update or insert.

For SQL engines that support RETURNING clauses, the `RETURNING` clause will be used to refresh field values, unless `sqoPreferRefresh` is specified in TSQLQuery.Options (1540)

See also: TField.Providerflags (1487), TSQLQuery.Options (1540)

60.24.43 TSQLQuery.IndexDefs

Synopsis: List of local index Definitions.

Declaration: `Property IndexDefs :`

Visibility: published

Access:

Description: List of local index Definitions.

See also: TCustomBufDataset.IndexDefs (1487)

60.24.44 TSQLQuery.Options

Synopsis: Options controlling the behaviour of the dataset.

Declaration: `Property Options :`

Visibility: published

Access:

Description: `Options` controls the behaviour of the dataset. The following options can be specified:

sqoKeepOpenOnCommit The default SQLDB behaviour is to close all datasets connected to a transaction when a transaction is committed or rolled back, which means that transactions must remain active as long as the dataset is open. This can create problems with locking of records etc. With this option set, the dataset will be kept open. Note that setting this option will cause SQLDB to fetch all records in the result set in memory.

sqoAutoApplyUpdates Setting this option will make `TSQLQuery` call `ApplyUpdates` after every `Post` or `Delete` operation.

sqoAutoCommit Setting this option will make `TSQLQuery` call commit after every `ApplyUpdates`

sqoCancelUpdatesOnRefresh Setting this option will cause `TSQLQuery` to abandon all pending changes when `Refresh` is called. The default behaviour is to raise an exception when `Refresh` is called and there are pending changes

sqoPreferRefresh If the database engine supports `RETURNING`, then the returning mechanism is used to fetch field values after an update of the database. Setting this option will disable the use of `RETURNING` and will fetch updated or new values instead with the TSQLQuery.RefreshSQL (1539) property or a constructed refresh SQL statement.

See also: TCustomSQLQuery.ApplyUpdates (1509), TCustomSQLQuery.Post (1509), TCustomSQLQuery.Delete (1509)

60.24.45 TSQLQuery.Params

Synopsis: Parameters detected in the SQL statement.

Declaration: `Property Params :`

Visibility: published

Access:

Description: `Params` contains the parameters used in the SQL statement. This collection is only updated when ParseSQL (1542) is `True`. For each named parameter in the SQL (1537) property, a named item will appear in the collection, and the collection will be used to retrieve values from.

When Open (515) or ExecSQL (1507) is called, and the Datasource (1543) property is not `Nil`, then for each parameter for which no value was explicitly set (its Bound (628) property is `False`), the value will be retrieved from the dataset connected to the datasource.

For each parameter, a field with the same name will be searched, and its value and type will be copied to the (unbound) parameter. The parameter remains unbound.

The Update, delete and insert SQL statements are not scanned for parameters.

See also: TSQLQuery.SQL (1537), TSQLQuery.ParseSQL (1542), TParam.Bound (628), UsingParams (1492), UpdateSQLS (1491)

60.24.46 TSQLQuery.ParamCheck

Synopsis: Should the SQL statement be checked for parameters.

Declaration: `Property ParamCheck :`

Visibility: published

Access:

Description: `ParamCheck` must be set to `False` to disable the parameter check. The default value `True` indicates that the SQL statement should be checked for parameter names (in the form `:ParamName`), and corresponding TParam (617) instances should be added to the Params (1540) property.

When executing some DDL statements, e.g. a "create procedure" SQL statement can contain parameters. These parameters should not be converted to `TParam` instances.

See also: TParam (617), Params (1540), ParamCheck (1541)

60.24.47 TSQLQuery.Macros

Synopsis: Set of macros for this SQL statement.

Declaration: `Property Macros :`

Visibility: published

Access:

Description: `Macros` is a collection of named macro values. In difference with Params (1540) the macro value is always replaced textually in the SQL statement before it is sent to the SQL engine. This allows you to parametrize parts of the SQL statement that the SQL engine will not let you parametrize: the table name, the order by clause or an IN clause in a SQL select statement. Macros are resolved before parameters are resolved.

See also: Params (1540)

60.24.48 TSQLQuery.MacroCheck

Synopsis: Check for macros in the SQL statement.

Declaration: `Property MacroCheck :`

Visibility: published

Access:

Description: `MacroC heck` is the macro equivalent of ParamCheck (1541): if set to `True`, it instructs the query component to check the SQL statement text for macros and add them to the Macros (1541) collection.

See also: ParamCheck (1541), Macros (1541)

60.24.49 TSQLQuery.MacroChar

Synopsis: Macro delimiter character.

Declaration: `Property MacroChar :`

Visibility: published

Access:

Description: `MacroChar` is the macro delimiter character. A macro is delimited by this character on both sides: start and end. The default is the % (percent) sign.

60.24.50 TSQLQuery.ParseSQL

Synopsis: Should the SQL statement be parsed or not.

Declaration: `Property ParseSQL :`

Visibility: published

Access:

Description: `ParseSQL` can be set to `False` to prevent `TSQLQuery` from parsing the SQL (1537) property and attempting to detect the statement type or updating the Params (1540) or StatementType (1532) properties.

This can be used when SQLDB has problems parsing the SQL statement, or when the SQL statement contains parameters that are part of a DDL statement such as a `CREATE PROCEDURE` statement to create a stored procedure.

Note that in this case the statement will be passed as-is to the SQL engine, no parameter values will be passed on.

See also: TSQLQuery.SQL (1537), TSQLQuery.Params (1540)

60.24.51 TSQLQuery.UpdateMode

Synopsis: How to create update SQL statements.

Declaration: `Property UpdateMode :`

Visibility: published

Access:

Description: `UpdateMode` determines how the `WHERE` clause of the UpdateSQL (1538) and DeleteSQL (1539) statements are auto-generated.

upWhereAll Use all old field values.

upWhereChanged Use only old field values of modified fields.

upWhereKeyOnly Only use key fields in the where clause.

See also: TSQLQuery.UpdateSQL (1538), TSQLQuery.InsertSQL (1538)

60.24.52 TSQLQuery.UsePrimaryKeyAsKey

Synopsis: Should primary key fields be marked `pfInKey`.

Declaration: `Property UsePrimaryKeyAsKey :`

Visibility: published

Access:

Description: `UsePrimaryKeyAsKey` can be set to `True` to let `TSQLQuery` fetch all server indexes and if there is a primary key, update the ProviderFlags (573) of the fields in the primary key with pfInKey (452).

The effect of this is that when UpdateMode (1542) equals `upWhereKeyOnly`, then only the fields that are part of the primary key of the table will be used in the update statements. For more information, see UpdateSQLs (1491).

Note that this property only takes effect if the fields are the default fields: if persistent fields were created, the providerflags of the fields are not updated.

See also: TSQLQuery.UpdateMode (1542), #fcl.bufdataset.TCustomBufDataset.Unidirectional (254), TField.ProviderFlags (573), pfInKey (452), UpdateSQLs (1491)

60.24.53 TSQLQuery.DataSource

Synopsis: Source for parameter values for unbound parameters.

Declaration: `Property DataSource :`

Visibility: published

Access:

Description: `Datasource` can be set to a dataset which will be used to retrieve values for the parameters if they were not explicitly specified.

When Open (515) or ExecSQL (1507) is called, and the `Datasource` property is not `Nil` then for each parameter for which no value was explicitly set (its Bound (628) property is `False`), the value will be retrieved from the dataset connected to the datasource.

For each parameter, a field with the same name will be searched, and its value and type will be copied to the (unbound) parameter. The parameter remains unbound.

See also: Params (1540), ExecSQL (1507), UsingParams (1492), TParam.Bound (628)

60.24.54 TSQLQuery.Sequence

Synopsis: Sequence to use for auto-generating values using a sequence.

Declaration: `Property Sequence :`

Visibility: published

Access:

Description: `Sequence` allows `TSQLQuery` to automate generation of a new value for a field using a sequence in the database.

To this end, the properties in TSQLSequence (1552) must be set to appropriate values, and `TSQLQuery` will automatically generate a new value for the indicated field during insert or post (depending on the value of TSQLSequence.ApplyEvent (1554)).

See also: TSQLSequence (1552)

60.24.55 TSQLQuery.ServerFilter

Synopsis: Append server-side filter to SQL statement.

Declaration: `Property ServerFilter :`

Visibility: published

Access:

Description: `ServerFilter` can be set to a valid `WHERE` clause (without the `WHERE` keyword). It will be appended to the `select` statement in SQL (1537), when ServerFiltered (1544) is set to `True`. if ServerFiltered (1544) is set to `False`, `ServerFilter` is ignored.

If the dataset is active and ServerFiltered (1544) is set to true, then changing this property will re-fetch the data from the server.

This property cannot be used when ParseSQL (1542) is `False`, because the statement must be parsed in order to know where the `WHERE` clause must be inserted: the `TSQLQuery` class will intelligently insert the clause in an SQL select statement.

Errors: Setting this property when ParseSQL (1542) is `False` will result in an exception.

See also: ServerFiltered (1544)

60.24.56 TSQLQuery.ServerFiltered

Synopsis: Should server-side filter be applied.

Declaration: `Property ServerFiltered :`

Visibility: published

Access:

Description: `ServerFiltered` can be set to `True` to apply ServerFilter (1544). A change in the value for this property will re-fetch the query results if the dataset is active.

Errors: Setting this property to `True` when ParseSQL (1542) is `False` will result in an exception.

See also: ParseSQL (1542), ServerFilter (1544)

60.24.57 TSQLQuery.ServerIndexDefs

Synopsis: List of indexes on the primary table of the query.

Declaration: `Property ServerIndexDefs :`

Visibility: published

Access:

Description: `ServerIndexDefs` will be filled - during the `Prepare` call - with the list of indexes defined on the primary table in the query if UsePrimaryKeyAsKey (1542) is `True`. If a primary key is found, then the fields in it will be marked

See also: UsePrimaryKeyAsKey (1542), Prepare (1507)

60.25 TSQLScript

60.25.1 Description

`TSQLScript` is a component that can be used to execute many SQL statements using a TSQLQuery (1529) component. The SQL statements are specified in a script TSQLScript.Script (1549) separated by a terminator character (typically a semicolon (;)).

See also: TSQLTransaction (1558), TSQLConnection (1514), TCustomSQLQuery.ExecSQL (1507), TSQLQuery.SQL (1537)

60.25.2 Method overview

Page	Method	Description
1545	Create	Create a new `TSQLScript` instance.
1546	Destroy	Remove the `TSQLScript` instance from memory.
1546	Execute	Execute the script.
1546	ExecuteScript	Convenience function, simply calls `Execute`.

60.25.3 Property overview

Page	Properties	Access	Description
1546	Aborted		True when the script was aborted.
1548	AutoCommit		Automatically commit every statement.
1550	CommentsinSQL		Should comments be passed to the SQL engine ?
1547	DataBase	rw	Database on which to execute the script.
1549	Defines		Defined macros.
1549	Directives		List of directives.
1548	DollarStrings		List of alternate string delimiter token sequences.
1547	Line		Current line of execution in the script.
1547	OnDirective	rw	Event handler if a directive is encountered.
1552	OnException		Exception handling event.
1549	Script		The script to execute.
1550	Terminator		Terminator character.
1547	Transaction	rw	Transaction to use in the script.
1551	UseCommit		Control automatic handling of the `COMMIT` command.
1551	UseDefines		Automatically handle pre-processor defines.
1548	UseDollarString		Enable support for dollarstrings.
1550	UseSetTerm		Should the SET TERM directive be recognized.

60.25.4 TSQLScript.Create

Synopsis: Create a new `TSQLScript` instance.

Declaration: `constructor Create(AOwner: TComponent); Override`

Visibility: public

Description: `Create` instantiates a TSQLQuery (1529) instance which will be used to execute the queries, and then calls the inherited constructor.

See also: TSQLScript.Destroy (1546)

60.25.5 TSQLScript.Destroy

Synopsis: Remove the `TSQLScript` instance from memory.

Declaration: `destructor Destroy; Override`

Visibility: public

Description: `Destroy` frees the TSQLQuery (1529) instance that was created during the `Create` constructor from memory and then calls the inherited destructor.

See also: TSQLScript.Create (1545)

60.25.6 TSQLScript.Execute

Synopsis: Execute the script.

Declaration: `procedure Execute; Override`

Visibility: public

Description: `Execute` will execute the statements specified in Script (1549) one by one, till the last statement is processed or an exception is raised.

If an error occurs during execution, normally an exception is raised. If the TSQLScript.OnException (1552) event handler is set, it may stop the event handler.

Errors: Handle errors using TSQLScript.OnException (1552).

See also: Script (1549), TSQLScript.OnException (1552)

60.25.7 TSQLScript.ExecuteScript

Synopsis: Convenience function, simply calls `Execute`.

Declaration: `procedure ExecuteScript`

Visibility: public

Description: `ExecuteScript` is a convenience function, it simply calls `Execute`. The statements in the script will be executed one by one.

60.25.8 TSQLScript.Aborted

Synopsis: True when the script was aborted.

Declaration: `Property Aborted :`

Visibility: public

Access:

Description: `Aborted` is set to `True` if the SQL script execution is aborted by one of the directives in the script. It is read-only.

60.25.9 TSQLScript.Line

Synopsis: Current line of execution in the script.

Declaration: `Property Line :`

Visibility: public

Access:

Description: `Line` is the line number (0 based) of the currently executed statement in the script. For multiline statements, the last line of the statement is counted as the current line.

60.25.10 TSQLScript.DataBase

Synopsis: Database on which to execute the script.

Declaration: `Property DataBase : TDatabase`

Visibility: published

Access: Read,Write

Description: `Database` should be set to the TSQLConnection (1514) descendent. All SQL statements in the Script (1549) property will be executed on this database.

See also: TSQLConnection (1514), TSQLScript.Transaction (1547), TSQLScript.Script (1549)

60.25.11 TSQLScript.Transaction

Synopsis: Transaction to use in the script.

Declaration: `Property Transaction : TDBTransaction`

Visibility: published

Access: Read,Write

Description: `Transaction` is the transaction instance to use when executing statements. If the SQL script contains any `COMMIT` statements, they will be handled using the TSQLTRansaction.CommitRetaining (1559) method.

See also: TSQLTransaction (1558), TSQLTransaction.CommitRetaining (1559), TSQLScript.Database (1547)

60.25.12 TSQLScript.OnDirective

Synopsis: Event handler if a directive is encountered.

Declaration: `Property OnDirective : TSQLScriptDirectiveEvent`

Visibility: published

Access: Read,Write

Description: `OnDirective` is called when a directive is encountered. When parsing the script, the script engine checks the first word of the statement. If it matches one of the words in Directives (1549) property then the `OnDirective` event handler is called with the name of the directive and the rest of the statement as parameters. This can be used to handle all kind of pre-processing actions such as `Set term \;`

See also: Directives (1549)

60.25.13 TSQLScript.AutoCommit

Synopsis: Automatically commit every statement.

Declaration: `Property AutoCommit :`

Visibility: published

Access:

Description: `AutoCommit` can be set to `True` to commit every executed statement in the script. By default, this is set to false.

See also: TSQLScript.Transaction (1547)

60.25.14 TSQLScript.UseDollarString

Synopsis: Enable support for dollarstrings.

Declaration: `Property UseDollarString :`

Visibility: published

Access:

Description: `UseDollarString` enables support for so-called "DollarString" delimiters for string literals. This means that the normal string literal delimiter (') is enhanced with any sring appearing in the DollarStrings (1548) property.

Setting `UseDollarString` to true incurs a speed penalty, so it is better not to enable it unless it is really necessary.

This is needed for instance for PostGreSQL, where stored procedure code blocks are enclosed in "$$" signs, and are treated as a string literal.

See also: TSQLScript.DollarStrings (1548)

60.25.15 TSQLScript.DollarStrings

Synopsis: List of alternate string delimiter token sequences.

Declaration: `Property DollarStrings :`

Visibility: published

Access:

Description: `DollarStrings` contains a list of additional string delimiter tokens. The value of this property is ignored unless TSQLScript.UseDollarString (1548) is also set to `True`.

For PostGreSQL, this should be set to $$, as this is the most commonly used string delimiter for stored procedures.

See also: TSQLScript.UseDollarString (1548)

60.25.16 TSQLScript.Directives

Synopsis: List of directives.

Declaration: `Property Directives :`

Visibility: published

Access:

Description: `Directives` is a stringlist with words that should be recognized as directives. They will be handled using the OnDirective (1547) event handler. The list should contain one word per line, no spaces allowed.

See also: OnDirective (1547)

60.25.17 TSQLScript.Defines

Synopsis: Defined macros.

Declaration: `Property Defines :`

Visibility: published

Access:

Description: `Defines` contains the list of defined macros for use with the TSQLScript.UseDefines (1551) property. Each line should contain a macro name. The names of the macros are case insensitive. The `#DEFINE` and `#UNDEFINE` directives will add or remove macro names from this list.

See also: TSQLScript.UseDefines (1551)

60.25.18 TSQLScript.Script

Synopsis: The script to execute.

Declaration: `Property Script :`

Visibility: published

Access:

Description: `Script` contains the list of SQL statements to be executed. The statements should be separated by the character specified in the Terminator (1550) property. Each of the statement will be executed on the database specified in Database (1547). using the equivalent of the TCustomSQLQuery.ExecSQL (1507) statement. The statements should not return result sets, but other than that all kind of statements are allowed.

Comments will be conserved and passed on in the statements to be executed, depending on the value of the TSQLScript.CommentsinSQL (1550) property. If that property is `False`, comments will be stripped prior to executing the SQL statements.

See also: TSQLScript.CommentsinSQL (1550), TSQLScript.Terminator (1550), TSQLScript.DataBase (1547)

60.25.19 TSQLScript.Terminator

Synopsis: Terminator character.

Declaration: `Property Terminator :`

Visibility: published

Access:

Description: `Terminator` is the character used by `TSQLScript` to delimit SQL statements. By default it equals the semicolon (;), which is the customary SQL command terminating character. By itself `TSQLScript` does not recognize complex statements such as `Create Procedure` which can contain terminator characters such as ";". Instead, `TSQLScript` will scan the script for the `Terminator` character. Using directives such as `SET TERM` the terminator character may be changed in the script.

See also: OnDirective (1547), Directives (1549)

60.25.20 TSQLScript.CommentsinSQL

Synopsis: Should comments be passed to the SQL engine ?

Declaration: `Property CommentsinSQL :`

Visibility: published

Access:

Description: `CommentsInSQL` can be set to `True` to let `TSQLScript` preserve any comments it finds in the script. The comments will be passed to the SQLConnection as part of the commands. If the property is set to `False` the comments are discarded.

By default, `TSQLScript` discards comments.

See also: TSQLScript.Script (1549)

60.25.21 TSQLScript.UseSetTerm

Synopsis: Should the SET TERM directive be recognized.

Declaration: `Property UseSetTerm :`

Visibility: published

Access:

Description: `UseSetTerm` can be set to `True` to let `TSQLScript` automatically handle the `SET TERM` directive and set the TSQLSCript.Terminator (1550) character based on the value specified in the `SET TERM` directive. This means that the following directive:

```
SET TERM ^ ;
```

will set the terminator to the caret character. Conversely, the

```
SET TERM ; ^
```

will then switch the terminator character back to the commonly used semicolon (;).

See also: TSQLSCript.Terminator (1550), TSQLSCript.Script (1549), TSQLSCript.Directives (1549)

60.25.22 TSQLScript.UseCommit

Synopsis: Control automatic handling of the COMMIT command.

Declaration: `Property UseCommit :`

Visibility: published

Access:

Description: `UseCommit` can be set to `True` to let `TSQLScript` automatically handle the commit command as a directive. If it is set, the `COMMIT` command is registered as a directive, and the TSQLScript.Transaction (1547) will be committed and restarted at once whenever the `COMMIT` directive appears in the script.

If this property is set to `False` then the commit command will be passed on to the SQL engine like any other SQL command in the script.

See also: TSQLScript.Transaction (1547), TSQLScript.Directives (1549)

60.25.23 TSQLScript.UseDefines

Synopsis: Automatically handle pre-processor defines.

Declaration: `Property UseDefines :`

Visibility: published

Access:

Description: `UseDefines` will automatically register the following pre-processing directives:

```
#IFDEF
#IFNDEF
#ELSE
#ENDIF
#DEFINE
#UNDEF
#UNDEFINE
```

Additionally, these directives will be automatically handled by the `TSQLScript` component. This can be used to add conditional execution of the SQL script: they are treated as the conditional compilation statements found in the C macro preprocessor or the FPC conditional compilation features. The initial list of defined macros can be specified in the Defines (1549) property, where one define per line can be specified.

In the following example, the correct statement to create a sequence is selected based on the presence of the macro `FIREBIRD` in the list of defines:

```
#IFDEF FIREBIRD
CREATE GENERATOR GEN_MYID;
#ELSE
CREATE SEQUENCE GEN_MYID;
#ENDIF
```

See also: TSQLScript.Script (1549), TSQLScript.Defines (1549)

60.25.24 TSQLScript.OnException

Synopsis: Exception handling event.

Declaration: `Property OnException :`

Visibility: published

Access:

Description: `OnException` can be set to handle an exception during the execution of a statement or directive when the script is executed. The exception is passed to the handler in the `TheException` parameter. On return, the value of the `Continue` parameter is checked: if it is set to `True`, then the exception is ignored. If it is set to `False` (the default), then the exception is re-raised, and script execution will stop.

See also: TSQLScript.Execute (1546)

60.26 TSQLSequence

60.26.1 Description

`TSQLSequence` is an auxiliary class, used to auto-generate numerical values for fields in databases that support sequences; it is used as a property of TSQLQuery (1529) and its properties determine which field must be auto-generated, and at what moment this value must be generated.

See also: TSQLConnection.GetSequenceNames (1518), TSQLConnection.GetNextValue (1519)

60.26.2 Method overview

Page	Method	Description
1553	Apply	Apply a new value to a field.
1553	Assign	Assign one `TSQLSequence` to another.
1552	Create	Create a new instance.
1553	GetNextValue	Get a next value for the sequence.

60.26.3 Property overview

Page	Properties	Access	Description
1554	ApplyEvent	rw	When to apply the new value.
1553	FieldName	rw	Field to apply sequence to.
1554	IncrementBy	rw	Value to increment sequence with.
1554	SequenceName	rw	Sequence name to get values from.

60.26.4 TSQLSequence.Create

Synopsis: Create a new instance.

Declaration: `constructor Create(AQuery: TCustomSQLQuery)`

Visibility: public

Description: `Create` instantiates a new sequence. It requires a TSQLQuery (1529) instance, which it needs to have access to a connection.

See also: TSQLQuery (1529)

60.26.5 TSQLSequence.Assign

Synopsis: Assign one `TSQLSequence` to another.

Declaration: `procedure Assign(Source: TPersistent); Override`

Visibility: public

Description: `Assign` is overridden by `TSQLSequence` to copy all properties from one instance to another.

Errors: None.

See also: TSQLSequence.FieldName (1553), TSQLSequence.SequenceName (1554), TSQLSequence.IncrementBy (1554)

60.26.6 TSQLSequence.Apply

Synopsis: Apply a new value to a field.

Declaration: `procedure Apply`

Visibility: public

Description: `Apply` applies the new value it gets for TSQLSequence.SequenceName (1554) using TSQLSequence.GetNextValue (1553) to the field TSQLSequence.FieldName (1553) of the dataset it is attached to.

Errors: If the dataset is not attached to a connected database, an exception will be raised.

See also: TSQLSequence.GetNextValue (1553), TSQLSequence.FieldName (1553), TSQLSequence.SequenceName (1554), TSQLSequence.IncrementBy (1554)

60.26.7 TSQLSequence.GetNextValue

Synopsis: Get a next value for the sequence.

Declaration: `function GetNextValue : Int64`

Visibility: public

Description: `GetNextValue` gets a new value for generator TSQLSequence.SequenceName (1554) using TSQLSequence.IncrementBy (1554)

Errors: If the dataset is not attached to a connected database, an exception will be raised.

See also: TSQLSequence.FieldName (1553), TSQLSequence.SequenceName (1554), TSQLSequence.IncrementBy (1554), TSQLSequence.Apply (1553), TSQLConnection.GetNextValue (1519)

60.26.8 TSQLSequence.FieldName

Synopsis: Field to apply sequence to.

Declaration: `Property FieldName : string`

Visibility: published

Access: Read,Write

Description: `FieldName` is the name of the field `TSQLSequence` will apply the new value to when Apply (1553) is called. It must be a valid fieldname of the dataset that owns the `TSQLSequence` instance.

See also: Apply (1553), SequenceName (1554), IncrementBy (1554)

60.26.9 TSQLSequence.SequenceName

Synopsis: Sequence name to get values from.

Declaration: `Property SequenceName : string`

Visibility: published

Access: Read,Write

Description: `SequenceName` is the name of the sequence `TSQLSequence` will get a new value of when GetNextValue (1553) is called.

See also: TSQLSequence.Apply (1553), TSQLSequence.GetNextValue (1553), TSQLSequence.IncrementBy (1554)

60.26.10 TSQLSequence.IncrementBy

Synopsis: Value to increment sequence with.

Declaration: `Property IncrementBy : Integer`

Visibility: published

Access: Read,Write

Description: `IncrementBy` is the value that will be added to the current value of the sequence TSQLSequence.SequenceName (1554) when TSQLSequence.GetNextValue (1553) is called.

See also: TSQLSequence.Apply (1553), TSQLSequence.GetNextValue (1553), TSQLSequence.SequenceName (1554)

60.26.11 TSQLSequence.ApplyEvent

Synopsis: When to apply the new value.

Declaration: `Property ApplyEvent : TSQLSequenceApplyEvent`

Visibility: published

Access: Read,Write

Description: `ApplyEvent` determines when the new value will be applied to a field: On new record (i.e. when Insert (512) or Append (503) is called) or when a newly inserted record is saved (when Post (515) is called).

See also: TSQLSequenceApplyEvent (1500)

60.27 TSQLStatement

60.27.1 Description

`TSQLStatement` is a descendent of TCustomSQLStatement (1510) which simply publishes the protected properties of that component.

See also: TCustomSQLStatement (1510)

60.27.2 Property overview

Page	Properties	Access	Description
1555	Database		Database instance to execute statement on.
1555	DataSource		Datasource to copy parameter values from.
1556	MacroCheck		Check for macros in the SQL statement.
1556	Macros		Set of macros for this SQL statement.
1555	ParamCheck		Should SQL be checked for parameters.
1556	Params		List of parameters.
1557	ParseSQL		Parse the SQL statement.
1557	SQL		The SQL statement to execute.
1557	Transaction		The transaction in which the SQL statement should be executed.

60.27.3 TSQLStatement.Database

Synopsis: Database instance to execute statement on.

Declaration: `Property Database :`

Visibility: published

Access:

Description: `Database` must be set to an instance of a TSQLConnection (1514) descendent. It must be set, together with Transaction (1557) in order to be able to call Prepare (1512) or Execute (1512).

See also: Transaction (1557), Prepare (1512), Execute (1512)

60.27.4 TSQLStatement.DataSource

Synopsis: Datasource to copy parameter values from.

Declaration: `Property DataSource :`

Visibility: published

Access:

Description: `Datasource` can be set to a #fcl.db.TDatasource (536) instance. When Execute (1512) is called, any unbound parameters remain empty, but if `DataSource` is set, the value of these parameters will be searched in the fields of the associated dataset. If a field with a name equal to the parameter is found, the value of that field is copied to the parameter. No such field exists, an exception is raised.

See also: #fcl.db.TDatasource (536), Execute (1512), #fcl.db.TParam.Bound (628)

60.27.5 TSQLStatement.ParamCheck

Synopsis: Should SQL be checked for parameters.

Declaration: `Property ParamCheck :`

Visibility: published

Access:

Description: `ParamCheck` must be set to `False` to disable the parameter check. The default value `True` indicates that the SQL statement should be checked for parameter names (in the form `:ParamName`), and corresponding TParam (617) instances should be added to the Params (1556) property.

When executing some DDL statements, e.g. a "create procedure" SQL statement can contain parameters. These parameters should not be converted to `TParam` instances.

See also: TParam (617), TSQLStatement.Params (1556), TSQLQuery.ParamCheck (1541)

60.27.6 TSQLStatement.Params

Synopsis: List of parameters.

Declaration: `Property Params :`

Visibility: published

Access:

Description: `Params` contains an item for each of the parameters in the SQL (1557) statement (in the form `:ParamName`). The collection is filled automatically if the ParamCheck (1555) property is `True`.

See also: SQL (1557), ParamCheck (1555), ParseSQL (1557)

60.27.7 TSQLStatement.MacroCheck

Synopsis: Check for macros in the SQL statement.

Declaration: `Property MacroCheck :`

Visibility: published

Access:

Description: `MacroCheck` is the macro equivalent of ParamCheck (1555): if set to `True`, it instructs the query component to check the SQL statement text for macros and add them to the Macros (1556) collection.

See also: ParamCheck (1555), Macros (1556)

60.27.8 TSQLStatement.Macros

Synopsis: Set of macros for this SQL statement.

Declaration: `Property Macros :`

Visibility: published

Access:

Description: `Macros` is a collection of named macro values. In difference with Params (1556) the macro value is always replaced textually in the SQL statement before it is sent to the SQL engine. This allows you to parametrize parts of the SQL statement that the SQL engine will not let you parametrize: the table name, the order by clause or an IN clause in a SQL select statement. Macros are resolved before parameters are resolved.

See also: Params (1556)

60.27.9 TSQLStatement.ParseSQL

Synopsis: Parse the SQL statement.

Declaration: `Property ParseSQL :`

Visibility: published

Access:

Description: `ParseSQL` can be set to `False` to disable parsing of the SQL (1557) property when it is set. The default behaviour (`ParseSQL=True`) is to parse the statement and detect what kind of SQL statement it is.

See also: SQL (1557), ParamCheck (1555)

60.27.10 TSQLStatement.SQL

Synopsis: The SQL statement to execute.

Declaration: `Property SQL :`

Visibility: published

Access:

Description: `SQL` must be set to the SQL statement to execute. It must not be a statement that returns a result set. This is the statement that will be passed on to the database engine when Prepare (1512) is called.

If ParamCheck (1555) equals `True` (the default), the SQL statement can contain parameter names where literal values can occur, in the form `:ParamName`. Keywords or table names cannot be specified as parameters. If the underlying database engine supports it, the parameter support of the database will be used to transfer the values from the Params (1556) collection. If not, it will be emulated. The `Params` collection is automatically populated when the SQL statement is set.

Some databases support executing multiple SQL statements in 1 call. Therefor, no attempt is done to ensure that `SQL` contains a single SQL statement. However, error reporting and the RowsAffected (1513) function may be wrong in such a case.

See also: ParseSQL (1557), CheckParams (1554), Params (1556), Prepare (1512), RowsAffected (1513)

60.27.11 TSQLStatement.Transaction

Synopsis: The transaction in which the SQL statement should be executed.

Declaration: `Property Transaction :`

Visibility: published

Access:

Description: `Transaction` should be set to a transaction connected to the instance of the database set in the Database (1555) property. This must be set before Prepare (1512) is called.

See also: Database (1555), Prepare (1512), TSQLTransaction (1558)

60.28 TSQLTransaction

60.28.1 Description

`TSQLTransaction` represents the transaction in which one or more TSQLQuery (1529) instances are doing their work. It contains the methods for committing or doing a rollback of the results of query. At least one `TSQLTransaction` must be used for each TSQLConnection (1514) used in an application.

See also: TSQLQuery (1529), TSQLConnection (1514)

60.28.2 Method overview

Page	Method	Description
1559	Commit	Commit the transaction, end transaction context.
1559	CommitRetaining	Commit the transaction, retain transaction context.
1558	Create	Create a new transaction.
1558	Destroy	Destroy transaction component.
1560	EndTransaction	End the transaction.
1559	Rollback	Roll back all changes made in the current transaction.
1560	RollbackRetaining	Roll back changes made in the transaction, keep transaction context.
1560	StartTransaction	Start a new transaction.

60.28.3 Property overview

Page	Properties	Access	Description
1561	Action	rw	Currently unused in SQLDB.
1561	Database		Database for which this component is handling connections.
1561	Handle	r	Low-level transaction handle.
1562	Options	rw	Transaction options.
1562	Params	rw	Transaction parameters.
1561	SQLConnection	rw	Database as TSQLConnection.

60.28.4 TSQLTransaction.Create

Synopsis: Create a new transaction.

Declaration: `constructor Create(AOwner: TComponent);  Override`

Visibility: public

Description: `Create` creates a new `TSQLTransaction` instance, but does not yet start a transaction context.

60.28.5 TSQLTransaction.Destroy

Synopsis: Destroy transaction component.

Declaration: `destructor Destroy;  Override`

Visibility: public

Description: `Destroy` will close all datasets connected to it, prior to removing the object from memory.

60.28.6 TSQLTransaction.Commit

Synopsis: Commit the transaction, end transaction context.

Declaration: `procedure Commit; Override`

Visibility: public

Description: `Commit` commits an active transaction. The changes will be irreversibly written to the database.

After this, the transaction is deactivated and must be reactivated with the StartTransaction (1560) method. To commit data while retaining an active transaction, execute CommitRetaining (1559) instead.

Errors: Executing `Commit` when no transaction is active will result in an exception. A transaction must be started by calling StartTransaction (1560). If the database backend reports an error, an exception is raised as well.

See also: StartTransaction (1560), CommitRetaining (1559), Rollback (1559), RollbackRetaining (1560)

60.28.7 TSQLTransaction.CommitRetaining

Synopsis: Commit the transaction, retain transaction context.

Declaration: `procedure CommitRetaining; Override`

Visibility: public

Description: `CommitRetaining` commits an active transaction. The changes will be irreversibly written to the database.

After this, the transaction is still active. To commit data and deactivate the transaction, execute Commit (1559) instead.

Errors: Executing `CommitRetaining` when no transaction is active will result in an exception. A transaction must be started by calling StartTransaction (1560). If the database backend reports an error, an exception is raised as well.

See also: StartTransaction (1560), Retaining (1559), Rollback (1559), RollbackRetaining (1560)

60.28.8 TSQLTransaction.Rollback

Synopsis: Roll back all changes made in the current transaction.

Declaration: `procedure Rollback; Override`

Visibility: public

Description: `Rollback` undoes all changes in the database since the start of the transaction. It can only be executed in an active transaction.

After this, the transaction is no longer active. To undo changes but keep an active transaction, execute RollbackRetaining (1560) instead.

Remark Changes posted in datasets that are coupled to this transaction will not be undone in memory: these datasets must be reloaded from the database (using `Close` and `Open` to reload the data as it is in the database.

Errors: Executing `Rollback` when no transaction is active will result in an exception. A transaction must be started by calling StartTransaction (1560). If the database backend reports an error, an exception is raised as well.

See also: StartTransaction (1560), CommitRetaining (1559), Commit (1559), RollbackRetaining (1560)

60.28.9 TSQLTransaction.RollbackRetaining

Synopsis: Roll back changes made in the transaction, keep transaction context.

Declaration: `procedure RollbackRetaining;  Override`

Visibility: public

Description: `RollbackRetaining` undoes all changes in the database since the start of the transaction. It can only be executed in an active transaction.

After this, the transaction is kept in an active state. To undo changes and close the transaction, execute Rollback (1559) instead.

Remark Changes posted in datasets that are coupled to this transaction will not be undone in memory: these datasets must be reloaded from the database (using `Close` and `Open` to reload the data as it is in the database.

Errors: Executing `RollbackRetaining` when no transaction is active will result in an exception. A transaction must be started by calling StartTransaction (1560). If the database backend reports an error, an exception is raised as well.

See also: StartTransaction (1560), Commit (1559), Rollback (1559), CommitRetaining (1559)

60.28.10 TSQLTransaction.StartTransaction

Synopsis: Start a new transaction.

Declaration: `procedure StartTransaction;  Override`

Visibility: public

Description: `StartTransaction` starts a new transaction context. All changes written to the database must be confirmed with a Commit (1559) or can be undone with a Rollback (1559) call.

Calling `StartTransaction` is equivalent to setting `Active` to `True`.

Errors: If `StartTransaction` is called while the transaction is still active, an exception will be raised.

See also: StartTransaction (1560), Commit (1559), Rollback (1559), CommitRetaining (1559), EndTransaction (1560)

60.28.11 TSQLTransaction.EndTransaction

Synopsis: End the transaction.

Declaration: `procedure EndTransaction;  Override`

Visibility: public

Description: `EndTransaction` is equivalent to RollBack (1559).

See also: RollBack (1559)

60.28.12 TSQLTransaction.Handle

Synopsis: Low-level transaction handle.

Declaration: `Property Handle : Pointer`

Visibility: public

Access: Read

Description: `Handle` is the low-level transaction handle object. It must not be used in application code. The actual type of this object depends on the type of TSQLConnection (1514) descendent.

60.28.13 TSQLTransaction.SQLConnection

Synopsis: Database as TSQLConnection.

Declaration: `Property SQLConnection : TSQLConnection`

Visibility: public

Access: Read,Write

Description: `SQLConnection` returns or sets the `Database` property, typecasted to `TSQLConnection`.

See also: TSQLConnection (1514)

60.28.14 TSQLTransaction.Action

Synopsis: Currently unused in SQLDB.

Declaration: `Property Action : TCommitRollbackAction`

Visibility: published

Access: Read,Write

Description: `Action` is currently unused in SQLDB.

60.28.15 TSQLTransaction.Database

Synopsis: Database for which this component is handling connections.

Declaration: `Property Database :`

Visibility: published

Access:

Description: `Database` should be set to the particular TSQLConnection (1514) instance this transaction is handling transactions in. All datasets connected to this transaction component must have the same value for their Database (1537) property.

See also: TSQLQuery.Database (1537), TSQLConnection (1514)

60.28.16 TSQLTransaction.Params

Synopsis: Transaction parameters.

Declaration: `Property Params : TStringList`

Visibility: published

Access: Read,Write

Description: `Params` can be used to set connection-specific parameters in the form of `Key=Value` pairs. The contents of this property therefor depends on the type of connection.

See also: TSQLConnection (1514)

60.28.17 TSQLTransaction.Options

Synopsis: Transaction options.

Declaration: `Property Options : TSQLTransactionOptions`

Visibility: published

Access: Read,Write

Description: `Options` can be used to control the behaviour of SQLDB for this transaction.

stoUseImplicit Use the implicit transaction support of the DB engine. This means that no explicit transaction start and stop commands will be sent to the server when the Commit or Rollback methods are called (effectively making them a no-op at the DB level).

stoExplicitStart When set, whenever an SQL statement is executed, the transaction must have been started explicitly. Default behaviour is that the TSQLStatement (1554) or TSQLQuery (1529) start the transaction as needed.

See also: TSQLStatement (1554), TSQLQuery (1529)

Chapter 61

Reference for unit 'SQLTypes'

61.1 Used units

Table 61.1: Used units by unit 'SQLTypes'

Name	Page
Classes	**??**
System	**??**
sysutils	**??**

61.2 Constants, types and variables

61.2.1 Types

```
TDBEventType = (detCustom,detPrepare,detExecute,detFetch,detCommit
  ,
                detRollBack,detParamValue,detActualSQL)
```

Table 61.2: Enumeration values for type TDBEventType

Value	Explanation
detActualSQL	Actual SQL as sent to engine message.
detCommit	Transaction Commit message.
detCustom	Custom event message.
detExecute	SQLExecute message.
detFetch	Fetch data message.
detParamValue	Parameter name and value message.
detPrepare	SQL prepare message.
detRollBack	Transaction rollback message.

`TDBEventType` describes the type of a database event message as generated by TSQLConnection (1514) through the OnLog (1522) event. event.

```
TDBEventTypes = Set of TDBEventType
```

TDBEventTypes is a set of TDBEventType (1563) values, which is used to filter the set of event messages that should be sent. The TSQLConnection.LogEvents (1523) property determines which events a particular connection will send.

```
TQuoteChars = Array[0..1] of Char
```

TQuoteChars is an array of characters that describes the used delimiters for string values.

```
TSchemaType = (stNoSchema,stTables,stSysTables,stProcedures,stColumns
  ,
                stProcedureParams,stIndexes,stPackages,stSchemata
  ,
                stSequences)
```

Table 61.3: Enumeration values for type TSchemaType

Value	Explanation
stColumns	Columns in a table.
stIndexes	Indexes for a table.
stNoSchema	No schema.
stPackages	Packages (for databases that support them).
stProcedureParams	Parameters for a stored procedure.
stProcedures	Stored procedures in database.
stSchemata	List of schemas in database(s) (for databases that support them).
stSequences	Sequences (for databases that support them).
stSysTables	System tables in database.
stTables	User Tables in database.

TSchemaType describes which schema information to retrieve in the TCustomSQLQuery.SetSchemaInfo (1508) call. Depending on its value, the result set of the dataset will have different fields, describing the requested schema data. The result data will always have the same structure.

```
TStatementType = (stUnknown,stSelect,stInsert,stUpdate,stDelete,stDDL
  ,
                  stGetSegment,stPutSegment,stExecProcedure,
  stStartTrans,stCommit,stRollback,stSelectForUpd)
```

Table 61.4: Enumeration values for type TStatementType

Value	Explanation
stCommit	The statement commits a transaction.
stDDL	The statement is a SQL DDL (Data Definition Language) statement.
stDelete	The statement is a SQL DELETE statement.
stExecProcedure	The statement executes a stored procedure.
stGetSegment	The statement is a SQL get segment statement.
stInsert	The statement is a SQL INSERT statement.
stPutSegment	The statement is a SQL put segment statement.
stRollback	The statement rolls back a transaction.
stSelect	The statement is a SQL SELECT statement.
stSelectForUpd	The statement selects data for update.
stStartTrans	The statement starts a transaction.
stUnknown	The statement type could not be detected.
stUpdate	The statement is a SQL UPDATE statement.

`TStatementType` describes the kind of SQL statement that was entered in the `SQL` property of a TSQLQuery (1529) component.

61.3 TSqlObjectIdenfier

61.3.1 Description

`TSqlObjectIdenfier` is a class that represents an SQL identifier in a database. It has 2 parts: the schema name and the object name.

See also: TSqlObjectIdenfierList (1563), #fcl.sqldb.TSQLConnection.GetObjectNames (1517)

61.3.2 Method overview

Page	Method	Description
1565	Create	Create a new instance of a `TSqlObjectIdenfier` class.
1566	FullName	Return the full name of the object.

61.3.3 Property overview

Page	Properties	Access	Description
1566	ObjectName	rw	Name of the object in the database.
1566	SchemaName	rw	Schema name.

61.3.4 TSqlObjectIdenfier.Create

Synopsis: Create a new instance of a `TSqlObjectIdenfier` class.

Declaration:
```
constructor Create(ACollection: TSqlObjectIdentifierList;
                   const AObjectName: string; const ASchemaName: string)
```

Visibility: public

Description: `Create` can be used to create a new `TSqlObjectIdenfier` instance and immediately set the values for the TSqlObjectIdenfier.ObjectName (1566) and TSqlObjectIdenfier.SchemaName (1566) properties.

See also: TSqlObjectIdenfier.ObjectName (1566), TSqlObjectIdenfier.SchemaName (1566)

61.3.5 TSqlObjectIdenfier.FullName

Synopsis: Return the full name of the object.

Declaration: `function FullName : string`

Visibility: public

Description: `FullName` returns the name of the object, prepended with the schema name if there is one. Both parts are separated by a dot (.) character.

See also: TSqlObjectIdenfier.SchemaName (1566), TSqlObjectIdenfier.ObjectName (1566)

61.3.6 TSqlObjectIdenfier.SchemaName

Synopsis: Schema name.

Declaration: `Property SchemaName : string`

Visibility: public

Access: Read,Write

Description: `SchemaName` is the name of the schema in which the object is defined. This is only set if the database actually supports schemas, for other databases, it is empty.

See also: TSqlObjectIdenfierList (1563), #fcl.sqldb.TSQLConnection.GetObjectNames (1517), TSqlObjectIdenfier.ObjectName (1566)

61.3.7 TSqlObjectIdenfier.ObjectName

Synopsis: Name of the object in the database.

Declaration: `Property ObjectName : string`

Visibility: public

Access: Read,Write

Description: `ObjectName` is the name of the object in the database. If the database supports schemas, then it must be combined with the SchemaName (1566) property in order to create a unique name for the object.

See also: TSqlObjectIdenfierList (1563), #fcl.sqldb.TSQLConnection.GetObjectNames (1517), TSqlObjectIdenfier.SchemaName (1566)

61.4 TSqlObjectIdentifierList

61.4.1 Method overview

Page	Method	Description
1567	AddIdentifier	

61.4.2 Property overview

Page	Properties	Access	Description
1567	Identifiers	rw	

61.4.3 TSqlObjectIdentifierList.AddIdentifier

Declaration:
```
function AddIdentifier : TSqlObjectIdenfier;  Overload
function AddIdentifier(const AObjectName: string;
                       const ASchemaName: string) : TSqlObjectIdenfier
                       ;  Overload
```

Visibility: public

61.4.4 TSqlObjectIdentifierList.Identifiers

Declaration:
```
Property Identifiers[Index: Integer]: TSqlObjectIdenfier; default
```

Visibility: public

Access: Read,Write

Chapter 62

Reference for unit 'streamcoll'

62.1 Used units

Table 62.1: Used units by unit 'streamcoll'

Name	Page
Classes	??
System	??
sysutils	??

62.2 Overview

The streamcoll unit contains the implementation of a collection (and corresponding collection item) which implements routines for saving or loading the collection to/from a stream. The collection item should implement 2 routines to implement the streaming; the streaming itself is not performed by the TStreamCollection (1571) collection item.

The streaming performed here is not compatible with the streaming implemented in the `Classes` unit for components. It is independent of the latter and can be used without a component to hold the collection.

The collection item introduces mostly protected methods, and the unit contains a lot of auxiliary routines which aid in streaming.

62.3 Procedures and functions

62.3.1 ColReadBoolean

Synopsis: Read a boolean value from a stream.

Declaration: `function ColReadBoolean(S: TStream) : Boolean`

Visibility: default

Description: `ColReadBoolean` reads a boolean from the stream `S` as it was written by ColWriteBoolean (1570) and returns the read value. The value cannot be read and written across systems that have different endian values.

See also: ColReadDateTime (1569), ColWriteBoolean (1570), ColReadString (1570), ColReadInteger (1569), ColReadFloat (1569), ColReadCurrency (1569)

62.3.2 ColReadCurrency

Synopsis: Read a currency value from the stream.

Declaration: `function ColReadCurrency(S: TStream) : Currency`

Visibility: default

Description: `ColReadCurrency` reads a currency value from the stream `S` as it was written by ColWriteCurrency (1570) and returns the read value. The value cannot be read and written across systems that have different endian values.

See also: ColReadDateTime (1569), ColReadBoolean (1568), ColReadString (1570), ColReadInteger (1569), ColReadFloat (1569), ColWriteCurrency (1570)

62.3.3 ColReadDateTime

Synopsis: Read a `TDateTime` value from a stream.

Declaration: `function ColReadDateTime(S: TStream) : TDateTime`

Visibility: default

Description: `ColReadDateTime` reads a currency value from the stream `S` as it was written by ColWriteDateTime (1570) and returns the read value. The value cannot be read and written across systems that have different endian values.

See also: ColWriteDateTime (1570), ColReadBoolean (1568), ColReadString (1570), ColReadInteger (1569), ColReadFloat (1569), ColReadCurrency (1569)

62.3.4 ColReadFloat

Synopsis: Read a floating point value from a stream.

Declaration: `function ColReadFloat(S: TStream) : Double`

Visibility: default

Description: `ColReadFloat` reads a double value from the stream `S` as it was written by ColWriteFloat (1571) and returns the read value. The value cannot be read and written across systems that have different endian values.

See also: ColReadDateTime (1569), ColReadBoolean (1568), ColReadString (1570), ColReadInteger (1569), ColWriteFloat (1571), ColReadCurrency (1569)

62.3.5 ColReadInteger

Synopsis: Read a 32-bit integer from a stream.

Declaration: `function ColReadInteger(S: TStream) : Integer`

Visibility: default

Description: `ColReadInteger` reads a 32-bit integer from the stream `S` as it was written by ColWriteInteger (1571) and returns the read value. The value cannot be read and written across systems that have different endian values.

See also: ColReadDateTime (1569), ColReadBoolean (1568), ColReadString (1570), ColWriteInteger (1571), ColReadFloat (1569), ColReadCurrency (1569)

62.3.6 ColReadString

Synopsis: Read a string from a stream.

Declaration: `function ColReadString(S: TStream) : string`

Visibility: default

Description: `ColReadStream` reads a string value from the stream `S` as it was written by ColWriteString (1571) and returns the read value. The value cannot be read and written across systems that have different endian values.

See also: ColReadDateTime (1569), ColReadBoolean (1568), ColWriteString (1571), ColReadInteger (1569), ColReadFloat (1569), ColReadCurrency (1569)

62.3.7 ColWriteBoolean

Synopsis: Write a boolean to a stream.

Declaration: `procedure ColWriteBoolean(S: TStream; AValue: Boolean)`

Visibility: default

Description: `ColWriteBoolean` writes the boolean `AValue` to the stream. `S`.

See also: ColReadBoolean (1568), ColWriteString (1571), ColWriteInteger (1571), ColWriteCurrency (1570), ColWriteDateTime (1570), ColWriteFloat (1571)

62.3.8 ColWriteCurrency

Synopsis: Write a currency value to stream.

Declaration: `procedure ColWriteCurrency(S: TStream; AValue: Currency)`

Visibility: default

Description: `ColWriteCurrency` writes the currency `AValue` to the stream `S`.

See also: ColWriteBoolean (1570), ColWriteString (1571), ColWriteInteger (1571), ColWriteDateTime (1570), ColWriteFloat (1571), ColReadCurrency (1569)

62.3.9 ColWriteDateTime

Synopsis: Write a `TDateTime` value to stream.

Declaration: `procedure ColWriteDateTime(S: TStream; AValue: TDateTime)`

Visibility: default

Description: `ColWriteDateTime` writes the `TDateTimeAValue` to the stream `S`.

See also: ColReadDateTime (1569), ColWriteBoolean (1570), ColWriteString (1571), ColWriteInteger (1571), ColWriteFloat (1571), ColWriteCurrency (1570)

62.3.10 ColWriteFloat

Synopsis: Write floating point value to stream.

Declaration: `procedure ColWriteFloat(S: TStream; AValue: Double)`

Visibility: default

Description: `ColWriteFloat` writes the double `AValue` to the stream `S`.

See also: ColWriteDateTime (1570), ColWriteBoolean (1570), ColWriteString (1571), ColWriteInteger (1571), ColReadFloat (1569), ColWriteCurrency (1570)

62.3.11 ColWriteInteger

Synopsis: Write a 32-bit integer to a stream.

Declaration: `procedure ColWriteInteger(S: TStream; AValue: Integer)`

Visibility: default

Description: `ColWriteInteger` writes the 32-bit integer `AValue` to the stream `S`. No endianness is observed.

See also: ColWriteBoolean (1570), ColWriteString (1571), ColReadInteger (1569), ColWriteCurrency (1570), ColWriteDateTime (1570)

62.3.12 ColWriteString

Synopsis: Write a string value to the stream.

Declaration: `procedure ColWriteString(S: TStream; const AValue: string)`

Visibility: default

Description: `ColWriteString` writes the string value `AValue` to the stream `S`.

See also: ColWriteBoolean (1570), ColReadString (1570), ColWriteInteger (1571), ColWriteCurrency (1570), ColWriteDateTime (1570), ColWriteFloat (1571)

62.4 EStreamColl

62.4.1 Description

Exception raised when an error occurs when streaming the collection.

62.5 TStreamCollection

62.5.1 Description

`TStreamCollection` is a TCollection (**??**) descendent which implements 2 calls LoadFromStream (1572) and SaveToStream (1572) which load and save the contents of the collection to a stream.

The collection items must be descendents of the TStreamCollectionItem (1573) class for the streaming to work correctly.

Note that the stream must be used to load collections of the same type.

See also: TStreamCollectionItem (1573)

62.5.2 Method overview

Page	Method	Description
1572	LoadFromStream	Load the collection from a stream.
1572	SaveToStream	Load the collection from the stream.

62.5.3 Property overview

Page	Properties	Access	Description
1572	Streaming	r	Indicates whether the collection is currently being written to stream.

62.5.4 TStreamCollection.LoadFromStream

Synopsis: Load the collection from a stream.

Declaration: `procedure LoadFromStream(S: TStream)`

Visibility: public

Description: `LoadFromStream` loads the collection from the stream `S`, if the collection was saved using SaveToStream (1572). It reads the number of items in the collection, and then creates and loads the items one by one from the stream.

Errors: An exception may be raised if the stream contains invalid data.

See also: TStreamCollection.SaveToStream (1572)

62.5.5 TStreamCollection.SaveToStream

Synopsis: Load the collection from the stream.

Declaration: `procedure SaveToStream(S: TStream)`

Visibility: public

Description: `SaveToStream` saves the collection to the stream `S` so it can be read from the stream with LoadFromStream (1572). It does this by writing the number of collection items to the stream, and then streaming all items in the collection by calling their `SaveToStream` method.

Errors: None.

See also: TStreamCollection.LoadFromStream (1572)

62.5.6 TStreamCollection.Streaming

Synopsis: Indicates whether the collection is currently being written to stream.

Declaration: `Property Streaming : Boolean`

Visibility: public

Access: Read

Description: `Streaming` is set to `True` if the collection is written to or loaded from stream, and is set again to `False` if the streaming process is finished.

See also: TStreamCollection.LoadFromStream (1572), TStreamCollection.SaveToStream (1572)

62.6 TStreamCollectionItem

62.6.1 Description

`TStreamCollectionItem` is a TCollectionItem (**??**) descendent which implements 2 abstract routines: `LoadFromStream` and `SaveToStream` which must be overridden in a descendent class.

These 2 routines will be called by the TStreamCollection (1571) to save or load the item from the stream.

See also: TStreamCollection (1571)

Chapter 63

Reference for unit 'streamex'

63.1 Used units

Table 63.1: Used units by unit 'streamex'

Name	Page
Classes	**??**
RtlConsts	**??**
System	**??**
sysutils	**??**

63.2 Overview

streamex implements some extensions to be used together with streams from the classes unit.

63.3 Constants, types and variables

63.3.1 Constants

`BUFFER_SIZE = 4096`

Default buffer size for `TStreamReader`.

`FILE_RIGHTS = 438`

Default file rights for `TStreamReader`.

`MIN_BUFFER_SIZE = 128`

Minimum buffer size for `TStreamReader`.

63.4 TBidirBinaryObjectReader

63.4.1 Description

`TBidirBinaryObjectReader` is a class descendent from TBinaryObjectReader (**??**), which implements the necessary support for BiDi data: the position in the stream (not available in the standard streaming) is emulated.

See also: TBidirBinaryObjectWriter (1575), TDelphiReader (1576)

63.4.2 Property overview

Page	Properties	Access	Description
1575	Position	rw	Position in the stream.

63.4.3 TBidirBinaryObjectReader.Position

Synopsis: Position in the stream.

Declaration: `Property Position : LongInt`

Visibility: public

Access: Read,Write

Description: `Position` exposes the position of the stream in the reader for use in the TDelphiReader (1576) class.

See also: TDelphiReader (1576)

63.5 TBidirBinaryObjectWriter

63.5.1 Description

`TBidirBinaryObjectReader` is a class descendent from TBinaryObjectWriter (**??**), which implements the necessary support for BiDi data.

See also: TBidirBinaryObjectWriter (1575), TDelphiWriter (1577)

63.5.2 Property overview

Page	Properties	Access	Description
1575	Position	rw	Position in the stream.

63.5.3 TBidirBinaryObjectWriter.Position

Synopsis: Position in the stream.

Declaration: `Property Position : LongInt`

Visibility: public

Access: Read,Write

Description: `Position` exposes the position of the stream in the writer for use in the TDelphiWriter (1577) class.

See also: TDelphiWriter (1577)

63.6 TDelphiReader

63.6.1 Description

`TDelphiReader` is a descendent of `TReader` which has support for BiDi Streaming. It overrides the stream reading methods for strings, and makes sure the stream can be positioned in the case of strings. For this purpose, it makes use of the TBidirBinaryObjectReader (1575) driver class.

See also: TDelphiWriter (1577), TBidirBinaryObjectReader (1575)

63.6.2 Method overview

Page	Method	Description
1576	GetDriver	Return the driver class as a TBidirBinaryObjectReader (1575) class.
1577	Read	Read data from stream.
1576	ReadStr	Overrides the standard `ReadStr` method.

63.6.3 Property overview

Page	Properties	Access	Description
1577	Position	rw	Position in the stream.

63.6.4 TDelphiReader.GetDriver

Synopsis: Return the driver class as a TBidirBinaryObjectReader (1575) class.

Declaration: `function GetDriver : TBidirBinaryObjectReader`

Visibility: public

Description: `GetDriver` simply returns the used driver and typecasts it as TBidirBinaryObjectReader (1575) class.

See also: TBidirBinaryObjectReader (1575)

63.6.5 TDelphiReader.ReadStr

Synopsis: Overrides the standard `ReadStr` method.

Declaration: `function ReadStr : AnsiString`

Visibility: public

Description: `ReadStr` makes sure the TBidirBinaryObjectReader (1575) methods are used, to store additional information about the stream position when reading the strings.

See also: TBidirBinaryObjectReader (1575)

63.6.6 TDelphiReader.Read

Synopsis: Read data from stream.

Declaration: `procedure Read(var Buf; Count: LongInt);  Override`

Visibility: public

Description: `Read` reads raw data from the stream. It reads `Count` bytes from the stream and places them in `Buf`. It forces the use of the TBidirBinaryObjectReader (1575) class when reading.

See also: TBidirBinaryObjectReader (1575), TDelphiReader.Position (1577)

63.6.7 TDelphiReader.Position

Synopsis: Position in the stream.

Declaration: `Property Position : LongInt`

Visibility: public

Access: Read,Write

Description: Position in the stream.

See also: TDelphiReader.Read (1577)

63.7 TDelphiWriter

63.7.1 Description

`TDelphiWriter` is a descendent of `TWriter` which has support for BiDi Streaming. It overrides the stream writing methods for strings, and makes sure the stream can be positioned in the case of strings. For this purpose, it makes use of the TBidirBinaryObjectWriter (1575) driver class.

See also: TDelphiReader (1576), TBidirBinaryObjectWriter (1575)

63.7.2 Method overview

Page	Method	Description
1578	FlushBuffer	Flushes the stream buffer.
1578	GetDriver	Return the driver class as a TBidirBinaryObjectWriter (1575) class.
1578	Write	Write raw data to the stream.
1578	WriteStr	Write a string to the stream.
1578	WriteValue	Write value type.

63.7.3 Property overview

Page	Properties	Access	Description
1579	Position	rw	Position in the stream.

63.7.4 TDelphiWriter.GetDriver

Synopsis: Return the driver class as a TBidirBinaryObjectWriter (1575) class.

Declaration: `function GetDriver : TBidirBinaryObjectWriter`

Visibility: public

Description: `GetDriver` simply returns the used driver and typecasts it as TBidirBinaryObjectWriter (1575) class.

See also: TBidirBinaryObjectWriter (1575)

63.7.5 TDelphiWriter.FlushBuffer

Synopsis: Flushes the stream buffer.

Declaration: `procedure FlushBuffer`

Visibility: public

Description: `FlushBuffer` flushes the internal buffer of the writer. It simply calls the `FlushBuffer` method of the driver class.

63.7.6 TDelphiWriter.Write

Synopsis: Write raw data to the stream.

Declaration: `procedure Write(const Buf; Count: LongInt);  Override`

Visibility: public

Description: `Write` writes `Count` bytes from `Buf` to the buffer, updating the position as needed.

63.7.7 TDelphiWriter.WriteStr

Synopsis: Write a string to the stream.

Declaration: `procedure WriteStr(const Value: Ansistring)`

Visibility: public

Description: `WriteStr` writes a string to the stream, forcing the use of the TBidirBinaryObjectWriter (1575) class methods, which update the position of the stream.

See also: TBidirBinaryObjectWriter (1575)

63.7.8 TDelphiWriter.WriteValue

Synopsis: Write value type.

Declaration: `procedure WriteValue(Value: TValueType)`

Visibility: public

Description: `WriteValue` overrides the same method in `TWriter` to force the use of the TBidirBinaryObjectWriter (1575) methods, which update the position of the stream.

See also: TBidirBinaryObjectWriter (1575)

63.7.9 TDelphiWriter.Position

Synopsis: Position in the stream.

Declaration:
```
Property Position : LongInt
```

Visibility: public

Access: Read,Write

Description: `Position` exposes the position in the stream as exposed by the TBidirBinaryObjectWriter (1575) instance used when streaming.

See also: TBidirBinaryObjectWriter (1575)

63.8 TFileReader

63.8.1 Description

`TFileReader` is a `TTextReader` descendent that takes a file on disk as the source of text data.

See also: TStreamReader (1585), TTextReader (1593)

63.8.2 Method overview

Page	Method	Description
1580	Close	Close the file.
1579	Create	Create a new instance of `TFileReader` for a disk file.
1580	Destroy	Remove the `TFileReader` instance from memory.
1580	ReadLine	Read a line of text.
1580	Reset	Reset the stream to its original position.

63.8.3 TFileReader.Create

Synopsis: Create a new instance of `TFileReader` for a disk file.

Declaration:
```
constructor Create(const AFileName: TFileName; AMode: Word;
                   ARights: Cardinal; ABufferSize: Integer);  Virtual
constructor Create(const AFileName: TFileName; AMode: Word;
                   ABufferSize: Integer);  Virtual
constructor Create(const AFileName: TFileName; ABufferSize: Integer)
                   ;  Virtual
constructor Create(const AFileName: TFileName);  Virtual
```

Visibility: public

Description: `Create` initializes a `TFileReader` using the provided `AFileName`. It will allocate a buffer of `ABufferSize` bytes for faster reading of data. If no buffer size is specified, BUFFER_SIZE (1574) will be used. If the `ABufferSize` argument is less than MIN_BUFFER_SIZE (1574), then `MIN_BUFFER_SIZE` bytes will be used.

The `AMode` can be used to specify the mode in which to open the file. This is one of the `fmOpenRead` and `fmShare*` constants which can be used in a `TFileStream` constructor. The file must be opened for reading.

Errors: If `AStream` is `Nil`, an #rtl.sysutils.EArgumentException (**??**) exception will be raised.

See also: TStreamReader.Destroy (1586), TFileStream (**??**), TFileStream.Create (**??**)

63.8.4 TFileReader.Destroy

Synopsis: Remove the `TFileReader` instance from memory.

Declaration: `destructor Destroy; Override`

Visibility: public

Description: `Destroy` closes the file and releases the buffer used to read data.

See also: TFileReader.Create (1579)

63.8.5 TFileReader.Reset

Synopsis: Reset the stream to its original position.

Declaration: `procedure Reset; Override`

Visibility: public

Description: `Reset` sets the stream to its original position. This is the stream-specific implementation of the abstract TTextReader.Reset (1594) method.

See also: TTextReader.Reset (1594)

63.8.6 TFileReader.Close

Synopsis: Close the file.

Declaration: `procedure Close; Override`

Visibility: public

Description: `Close` closes the text file. Any read operations after `Close` is called will fail. This is the stream-specific implementation of the abstract TTextReader.Close (1594)

See also: TTextReader.Close (1594)

63.8.7 TFileReader.ReadLine

Synopsis: Read a line of text.

Declaration: `procedure ReadLine(out AString: AnsiString); Override; Overload`

Visibility: public

Description: `ReadLine` will read a line of text from the text data source. A line of text is delimited by a CRLF character pair, a LF character or a CR character. The line ending characters are not included in the string.

The method exists in 2 versions: one function where the line of text is returned as the function result, one procedure where the line of text is returned in the `AString` parameter.

This is the `TFileReader` specific implementation of the abstract TTextReader.ReadLine (1594) method.

See also: Eof (1595), TTextReader.ReadLine (1594)

63.9 TStreamHelper

63.9.1 Description

`TStreamHelper` is a TStream (**??**) helper class which introduces some helper routines to read/write multi-byte integer values in a way that is endianness-safe.

See also: TStream (**??**)

63.9.2 Method overview

Page	Method	Description
1584	ReadDouble	Read a double-precision floating point value from the stream.
1583	ReadDWordBE	Read a DWord from the stream, big endian.
1581	ReadDWordLE	Read a DWord from the stream, little endian.
1583	ReadQWordBE	Read a QWord from the stream, big endian.
1582	ReadQWordLE	Read a QWord from the stream, little endian.
1584	ReadSingle	Read a single-precision floating point value from the stream.
1583	ReadWordBE	Read a Word from the stream, big endian.
1581	ReadWordLE	Read a Word from the stream, little endian.
1585	WriteDouble	Write a double-precision floating point value to the stream.
1584	WriteDWordBE	Write a DWord value, big endian.
1582	WriteDWordLE	Write a DWord value, little endian.
1584	WriteQWordBE	Write a QWord value, big endian.
1582	WriteQWordLE	Write a QWord value, little endian.
1585	WriteSingle	Write a single-precision floating point value to the stream.
1583	WriteWordBE	Write a word value, big endian.
1582	WriteWordLE	Write a word value, little endian.

63.9.3 TStreamHelper.ReadWordLE

Synopsis: Read a Word from the stream, little endian.

Declaration: `function ReadWordLE : Word`

Visibility: default

Description: `ReadWordLE` reads a word from the stream, little-endian (LSB first).

Errors: If not enough data is available an `EReadError` exception is raised.

See also: TStreamHelper.ReadDWordLE (1581), TStreamHelper.ReadQWordLE (1582), TStreamHelper.WriteWordLE (1582)

63.9.4 TStreamHelper.ReadDWordLE

Synopsis: Read a DWord from the stream, little endian.

Declaration: `function ReadDWordLE : dword`

Visibility: default

Description: `ReadWordLE` reads a DWord from the stream, little-endian (LSB first).

Errors: If not enough data is available an `EReadError` exception is raised.

See also: TStreamHelper.ReadWordLE (1581), TStreamHelper.ReadQWordLE (1582), TStreamHelper.WriteDWordLE (1582)

63.9.5 TStreamHelper.ReadQWordLE

Synopsis: Read a QWord from the stream, little endian.

Declaration: `function ReadQWordLE : QWord`

Visibility: default

Description: `ReadWordLE` reads a QWord from the stream, little-endian (LSB first).

Errors: If not enough data is available an `EReadError` exception is raised.

See also: TStreamHelper.ReadWordLE (1581), TStreamHelper.ReadDWordLE (1581), TStreamHelper.WriteQWordLE (1582)

63.9.6 TStreamHelper.WriteWordLE

Synopsis: Write a word value, little endian.

Declaration: `procedure WriteWordLE(w: Word)`

Visibility: default

Description: `WriteWordLE` writes a Word-sized value to the stream, little-endian (LSB first).

Errors: If not all data (2 bytes) can be written, an `EWriteError` exception is raised.

See also: TStreamHelper.ReadWordLE (1581), TStreamHelper.WriteDWordLE (1582), TStreamHelper.WriteQWordLE (1582)

63.9.7 TStreamHelper.WriteDWordLE

Synopsis: Write a DWord value, little endian.

Declaration: `procedure WriteDWordLE(dw: dword)`

Visibility: default

Description: `WriteDWordLE` writes a DWord-sized value to the stream, little-endian (LSB first).

Errors: If not all data (4 bytes) can be written, an `EWriteError` exception is raised.

See also: TStreamHelper.ReadDWordLE (1581), TStreamHelper.WriteWordLE (1582), TStreamHelper.WriteQWordLE (1582)

63.9.8 TStreamHelper.WriteQWordLE

Synopsis: Write a QWord value, little endian.

Declaration: `procedure WriteQWordLE(dq: QWord)`

Visibility: default

Description: `WriteQWordLE` writes a QWord-sized value to the stream, little-endian (LSB first).

Errors: If not all data (8 bytes) can be written, an `EWriteError` exception is raised.

See also: TStreamHelper.ReadQWordLE (1582), TStreamHelper.WriteDWordLE (1582), TStreamHelper.WriteWordLE (1582)

63.9.9 TStreamHelper.ReadWordBE

Synopsis: Read a Word from the stream, big endian.

Declaration: `function ReadWordBE : Word`

Visibility: default

Description: `ReadWordBE` reads a word from the stream, big-endian (MSB first).

Errors: If not enough data is available an `EReadError` exception is raised.

See also: TStreamHelper.ReadDWordBE (1583), TStreamHelper.ReadQWordBE (1583), TStreamHelper.WriteWordBE (1583)

63.9.10 TStreamHelper.ReadDWordBE

Synopsis: Read a DWord from the stream, big endian.

Declaration: `function ReadDWordBE : dword`

Visibility: default

Description: `ReadWordBE` reads a DWord from the stream, big-endian (MSB first).

Errors: If not enough data is available an `EReadError` exception is raised.

See also: TStreamHelper.ReadWordBE (1583), TStreamHelper.ReadQWordBE (1583), TStreamHelper.WriteDWordBE (1584)

63.9.11 TStreamHelper.ReadQWordBE

Synopsis: Read a QWord from the stream, big endian.

Declaration: `function ReadQWordBE : QWord`

Visibility: default

Description: `ReadWordBE` reads a QWord from the stream, big-endian (MSB first).

Errors: If not enough data is available an `EReadError` exception is raised.

See also: TStreamHelper.ReadWordBE (1583), TStreamHelper.ReadDWordBE (1583), TStreamHelper.WriteQWordBE (1584)

63.9.12 TStreamHelper.WriteWordBE

Synopsis: Write a word value, big endian.

Declaration: `procedure WriteWordBE(w: Word)`

Visibility: default

Description: `WriteWordBE` writes a Word-sized value to the stream, big-endian (MSB first).

Errors: If not all data (2 bytes) can be written, an `EWriteError` exception is raised.

See also: TStreamHelper.ReadWordBE (1583), TStreamHelper.WriteDWordBE (1584), TStreamHelper.WriteQWordBE (1584)

63.9.13 TStreamHelper.WriteDWordBE

Synopsis: Write a DWord value, big endian.

Declaration: `procedure WriteDWordBE(dw: dword)`

Visibility: default

Description: `WriteDWordBE` writes a DWord-sized value to the stream, big-endian (MSB first).

Errors: If not all data (4 bytes) can be written, an `EWriteError` exception is raised.

See also: TStreamHelper.ReadDWordBE (1583), TStreamHelper.WriteWordBE (1583), TStreamHelper.WriteQWordBE (1584)

63.9.14 TStreamHelper.WriteQWordBE

Synopsis: Write a QWord value, big endian.

Declaration: `procedure WriteQWordBE(dq: QWord)`

Visibility: default

Description: `WriteQWordBE` writes a QWord-sized value to the stream, big-endian (MSB first).

Errors: If not all data (8 bytes) can be written, an `EWriteError` exception is raised.

See also: TStreamHelper.ReadQWordBE (1583), TStreamHelper.WriteDWordBE (1584), TStreamHelper.WriteWordBE (1583)

63.9.15 TStreamHelper.ReadSingle

Synopsis: Read a single-precision floating point value from the stream.

Declaration: `function ReadSingle : Single`

Visibility: default

Description: `ReadSingle` reads a single-precision floating point value from the stream and returns the value. No endianness corrections are performed.

Errors: If the end of stream is reached before all necessary bytes can be read, an EReadError (**??**) exception is raised.

See also: TStreamHelper.ReadDouble (1584), TStreamHelper.WriteSingle (1585)

63.9.16 TStreamHelper.ReadDouble

Synopsis: Read a double-precision floating point value from the stream.

Declaration: `function ReadDouble : Double`

Visibility: default

Description: `ReadDouble` reads a double-precision floating point value from the stream and returns the value. No endianness corrections are performed.

Errors: If the end of stream is reached before all necessary bytes can be read, an EReadError (**??**) exception is raised.

See also: TStreamHelper.ReadSingle (1584), TStreamHelper.WriteDouble (1585)

63.9.17 TStreamHelper.WriteSingle

Synopsis: Write a single-precision floating point value to the stream.

Declaration: `procedure WriteSingle(s: Single)`

Visibility: default

Description: `WriteSingle` writes the single-precision floating point value `S` to the stream. No endianness corrections are performed.

Errors: If not all bytes can be written, an EWriteError (**??**) exception is raised.

See also: TStreamHelper.ReadSingle (1584), TStreamHelper.WriteDouble (1585)

63.9.18 TStreamHelper.WriteDouble

Synopsis: Write a double-precision floating point value to the stream.

Declaration: `procedure WriteDouble(d: Double)`

Visibility: default

Description: `WriteDouble` writes the double-precision floating point value `D` to the stream. No endianness corrections are performed.

Errors: If not all bytes can be written, an EWriteError (**??**) exception is raised.

See also: TStreamHelper.ReadDouble (1584), TStreamHelper.WriteSingle (1585)

63.10 TStreamReader

63.10.1 Description

`TStreamReader` is a `TTextReader` descendent that takes a stream as the source of text data. It can free the stream and the buffer size to use for reading data can be set.

See also: TTextReader (1593)

63.10.2 Method overview

Page	Method	Description
1587	Close	Close and possibly free the stream.
1586	Create	Create a new instance of `TStreamReader` from a stream.
1586	Destroy	Destroy the `TStreamReader` instance.
1587	ReadLine	Read a line of text.
1586	Reset	Reset the stream to its original position.

63.10.3 Property overview

Page	Properties	Access	Description
1587	BaseStream	r	The stream with the text data.
1587	OwnsStream	rw	Should the stream be freed on close.

63.10.4 TStreamReader.Create

Synopsis: Create a new instance of `TStreamReader` from a stream.

Declaration:
```
constructor Create(AStream: TStream; ABufferSize: Integer;
                   AOwnsStream: Boolean);  Virtual
constructor Create(AStream: TStream);  Virtual
constructor Create(const aFilename: string)
constructor Create(const aFilename: string; aDetectBOM: Boolean)
constructor Create(const aFilename: string; aEncoding: TEncoding;
                   aDetectBOM: Boolean; aBufferSize: Integer);  Overload
```

Visibility: public

Description: `Create` initializes a `TStreamReader` using the provided `AStream`. It will allocate a buffer of `ABufferSize` bytes for faster reading of data. If no buffer size is specified, BUFFER_SIZE (1574) will be used. If the `ABufferSize` argument is less than MIN_BUFFER_SIZE (1574), then `MIN_BUFFER_SIZE` bytes will be used.

If `AOwnsStream` is true, the stream will be freed when the `TStreamReader` instance is freed. If omitted, its value is assumed to be `False`.

Errors: If `AStream` is `Nil`, an #rtl.sysutils.EArgumentException (**??**) exception will be raised.

See also: TStreamReader.Destroy (1586), TStream (**??**)

63.10.5 TStreamReader.Destroy

Synopsis: Destroy the `TStreamReader` instance.

Declaration:
```
destructor Destroy;  Override
```

Visibility: public

Description: `Destroy` frees the resources taken by the buffer, and frees the source stream (BaseStream (1587)) if `OwnsStream` is `True`.

See also: TStreamReader.OwnsStream (1587), TStreamReader.Create (1586), TStreamReader.BaseStream (1587)

63.10.6 TStreamReader.Reset

Synopsis: Reset the stream to its original position.

Declaration:
```
procedure Reset;  Override
```

Visibility: public

Description: `Reset` sets the stream to its original position. This is the stream-specific implementation of the abstract TTextReader.Reset (1594) method.

Errors: If the source stream (TStreamReader.BaseStream (1587)) is not seekable, then this method may raise an exception.

See also: TStreamReader.BaseStream (1587)

63.10.7 TStreamReader.Close

Synopsis: Close and possibly free the stream.

Declaration: `procedure Close;  Override`

Visibility: public

Description: `Close` closed the text data stream. It will free the source stream if `OwnsStream` is `True`

See also: TStreamReader.BaseStream (1587), TStreamReader.OwnsStream (1587)

63.10.8 TStreamReader.ReadLine

Synopsis: Read a line of text.

Declaration: `procedure ReadLine(out AString: Ansistring);  Override;  Overload`

Visibility: public

Description: `ReadLine` will read a line of text from the text data source. A line of text is delimited by a CRLF character pair, a LF character or a CR character. The line ending characters are not included in the string.

The method exists in 2 versions: one function where the line of text is returned as the function result, one procedure where the line of text is returned in the `AString` parameter.

This is the `TStreamReader` specific implementation of the abstract TTextReader.ReadLine (1594) method.

See also: Eof (1595), TTextReader.ReadLine (1594)

63.10.9 TStreamReader.BaseStream

Synopsis: The stream with the text data.

Declaration: `Property BaseStream : TStream`

Visibility: public

Access: Read

Description: `BaseStream` is the stream that was passed to the `TStreamReader` instance in the TStreamReader.Create (1586) call.

Manipulating the stream between calls to TStreamReader.ReadLine (1587) is not allowed, it will lead to wrong data being read from the stream.

See also: TStreamReader.Create (1586), TStreamReader.OwnsStream (1587)

63.10.10 TStreamReader.OwnsStream

Synopsis: Should the stream be freed on close.

Declaration: `Property OwnsStream : Boolean`

Visibility: public

Access: Read,Write

Description: `OwnsStream` determines whether the stream TStreamReader.BaseStream (1587) must be freed when TStreamReader.Close (1587) is called or when the `TStreamReader` instance is destroyed.

See also: TStreamReader.BaseStream (1587), TStreamReader.Close (1587)

63.11 TStreamWriter

63.11.1 Method overview

Page	Method	Description
1588	Close	
1588	Create	
1588	Destroy	
1588	Flush	
1589	OwnStream	
1589	Write	
1589	WriteLine	

63.11.2 Property overview

Page	Properties	Access	Description
1589	AutoFlush	rw	
1590	BaseStream	r	
1590	Encoding	r	
1590	NewLine	rw	

63.11.3 TStreamWriter.Create

Declaration:
```
constructor Create(aStream: TStream);  Overload
constructor Create(aStream: TStream; aEncoding: TEncoding;
                   aBufferSize: Integer);  Overload
constructor Create(const aFilename: string; aAppend: Boolean);  Overload
constructor Create(const aFilename: string; aAppend: Boolean;
                   aEncoding: TEncoding; aBufferSize: Integer);  Overload
```

Visibility: public

63.11.4 TStreamWriter.Destroy

Declaration:
```
destructor Destroy;  Override
```

Visibility: public

63.11.5 TStreamWriter.Close

Declaration:
```
procedure Close;  Override
```

Visibility: public

63.11.6 TStreamWriter.Flush

Declaration:
```
procedure Flush;  Override
```

Visibility: public

63.11.7 TStreamWriter.OwnStream

Declaration: `procedure OwnStream`

Visibility: public

63.11.8 TStreamWriter.Write

Declaration:
```
procedure Write(aValue: Boolean);  Override
procedure Write(aValue: Char);  Override
procedure Write(const aValue: TCharArray);  Override
procedure Write(aValue: Double);  Override
procedure Write(aValue: Integer);  Override
procedure Write(aValue: Int64);  Override
procedure Write(aValue: TObject);  Override
procedure Write(aValue: Single);  Override
procedure Write(const aValue: string);  Override
procedure Write(aValue: Cardinal);  Override
procedure Write(aValue: UInt64);  Override
procedure Write(const Fmt: string; aArgs: Array of const);  Override
procedure Write(const aValue: TCharArray; aIndex: Integer;
              aCount: Integer);  Override
```

Visibility: public

63.11.9 TStreamWriter.WriteLine

Declaration:
```
procedure WriteLine;  Override
procedure WriteLine(aValue: Boolean);  Override
procedure WriteLine(aValue: Char);  Override
procedure WriteLine(const aValue: TCharArray);  Override
procedure WriteLine(aValue: Double);  Override
procedure WriteLine(aValue: Integer);  Override
procedure WriteLine(aValue: Int64);  Override
procedure WriteLine(aValue: TObject);  Override
procedure WriteLine(aValue: Single);  Override
procedure WriteLine(const aValue: string);  Override
procedure WriteLine(aValue: Cardinal);  Override
procedure WriteLine(aValue: UInt64);  Override
procedure WriteLine(const Fmt: string; Args: Array of const);  Override
procedure WriteLine(const aValue: TCharArray; aIndex: Integer;
                  aCount: Integer);  Override
```

Visibility: public

63.11.10 TStreamWriter.AutoFlush

Declaration: `Property AutoFlush : Boolean`

Visibility: public

Access: Read,Write

63.11.11 TStreamWriter.NewLine

Declaration: `Property NewLine : string`

Visibility: public

Access: Read,Write

63.11.12 TStreamWriter.Encoding

Declaration: `Property Encoding : TEncoding`

Visibility: public

Access: Read

63.11.13 TStreamWriter.BaseStream

Declaration: `Property BaseStream : TStream`

Visibility: public

Access: Read

63.12 TStringReader

63.12.1 Description

`TStreamReader` is a `TTextReader` descendent that takes a single string as the source of text data.

See also: TTextReader (1593)

63.12.2 Method overview

Page	Method	Description
1591	Close	Close and possibly free the stream.
1590	Create	Create a new instance of `TStreamReader` from a string.
1591	Destroy	Free the `TStringReader` instance.
1591	ReadLine	Read a line of text.
1591	Reset	Reset the stream to its original position.

63.12.3 TStringReader.Create

Synopsis: Create a new instance of `TStreamReader` from a string.

Declaration:
```
constructor Create(const AString: AnsiString; ABufferSize: Integer)
                    ;  Virtual
constructor Create(const AString: AnsiString);  Virtual
```

Visibility: public

Description: `Create` initializes a `TStringReader` instance using the provided `AString`. It will allocate a buffer of `ABufferSize` bytes for faster reading of data. If no buffer size is specified, BUFFER_SIZE (1574) will be used. If the `ABufferSize` argument is less than MIN_BUFFER_SIZE (1574), then `MIN_BUFFER_SIZE` bytes will be used.

Errors: If `AStream` is `Nil`, an #rtl.sysutils.EArgumentException (**??**) exception will be raised.

See also: TStreamReader.Destroy (1586), TStream (**??**)

63.12.4 TStringReader.Destroy

Synopsis: Free the `TStringReader` instance.

Declaration: `destructor Destroy; Override`

Visibility: public

Description: `Destroy` deallocates all resources for the `TStringReader` instance.

See also: TStringReader.Create (1590)

63.12.5 TStringReader.Reset

Synopsis: Reset the stream to its original position.

Declaration: `procedure Reset; Override`

Visibility: public

Description: `Reset` sets the stream to its original position. This is the string-specific implementation of the abstract TTextReader.Reset (1594) method.

See also: TStreamReader.BaseStream (1587)

63.12.6 TStringReader.Close

Synopsis: Close and possibly free the stream.

Declaration: `procedure Close; Override`

Visibility: public

Description: `Close` closes the text reader. This is the string-specific implementation of the abstract TTextReader.Close (1594) method.

See also: TTextReader.Close (1594)

63.12.7 TStringReader.ReadLine

Synopsis: Read a line of text.

Declaration: `procedure ReadLine(out AString: AnsiString); Override; Overload`

Visibility: public

Description: `ReadLine` will read a line of text from the text data source. A line of text is delimited by a CRLF character pair, a LF character or a CR character. The line ending characters are not included in the string.

The method exists in 2 versions: one function where the line of text is returned as the function result, one procedure where the line of text is returned in the `AString` parameter.

This is the `TStringReader` specific implementation of the abstract TTextReader.ReadLine (1594) method.

See also: Eof (1595), TTextReader.ReadLine (1594)

63.13 TStringWriter

63.13.1 Method overview

Page	Method	Description
1592	Close	
1592	Create	
1592	Destroy	
1592	Flush	
1593	ToString	
1592	Write	
1593	WriteLine	

63.13.2 TStringWriter.Create

Declaration:
```
constructor Create;  Overload
constructor Create(aBuilder: TStringBuilder);  Overload
```

Visibility: public

63.13.3 TStringWriter.Destroy

Declaration:
```
destructor Destroy;  Override
```

Visibility: public

63.13.4 TStringWriter.Close

Declaration:
```
procedure Close;  Override
```

Visibility: public

63.13.5 TStringWriter.Flush

Declaration:
```
procedure Flush;  Override
```

Visibility: public

63.13.6 TStringWriter.Write

Declaration:
```
procedure Write(aValue: Boolean);  Override
procedure Write(aValue: Char);  Override
procedure Write(aValue: Char; aCount: Integer);  Override
procedure Write(const aValue: TCharArray);  Override
procedure Write(aValue: Double);  Override
procedure Write(aValue: Integer);  Override
procedure Write(aValue: Int64);  Override
procedure Write(aValue: TObject);  Override
procedure Write(aValue: Single);  Override
procedure Write(const aValue: string);  Override
procedure Write(aValue: Cardinal);  Override
procedure Write(aValue: QWord);  Override
procedure Write(const aFmt: string; aArgs: Array of const);  Override
```

```
procedure Write(const aValue: TCharArray; aIndex: Integer;
                aCount: Integer);  Override
```

Visibility: public

63.13.7 TStringWriter.WriteLine

Declaration:
```
procedure WriteLine;  Override
procedure WriteLine(aValue: Boolean);  Override
procedure WriteLine(aValue: Char);  Override
procedure WriteLine(const aValue: TCharArray);  Override
procedure WriteLine(aValue: Double);  Override
procedure WriteLine(aValue: Integer);  Override
procedure WriteLine(aValue: Int64);  Override
procedure WriteLine(aValue: TObject);  Override
procedure WriteLine(aValue: Single);  Override
procedure WriteLine(const aValue: string);  Override
procedure WriteLine(aValue: Cardinal);  Override
procedure WriteLine(aValue: UInt64);  Override
procedure WriteLine(const aFmt: string; aArgs: Array of const)
                    ;  Override
procedure WriteLine(const aValue: TCharArray; aIndex: Integer;
                    aCount: Integer);  Override
```

Visibility: public

63.13.8 TStringWriter.ToString

Declaration:
```
function ToString : string;  Override
```

Visibility: public

63.14 TTextReader

63.14.1 Description

`TTextReader` is an abstract class that provides a line-oriented reading API. It allows to read data from streams or memory blocks as if one was using regular pascal `Read` or `ReadLn` operations: the ReadLine (1594) procedure. Several descendents of this class exist which implement the reader interface for several sources of text data: TStreamReader (1585), TFileReader (1579), TStringReader (1590).

See also: TStreamReader (1585), TFileReader (1579), TStringReader (1590), ReadLine (1594)

63.14.2 Method overview

Page	Method	Description
1594	Close	Close the text data stream.
1594	Create	Instantiate a new instance.
1594	ReadLine	Read a line of text.
1594	Reset	Reset the reader to the start position.

63.14.3 Property overview

Page	Properties	Access	Description
1595	EndOfStream	r	
1595	Eof	r	Check whether the end of the text data is returned.

63.14.4 TTextReader.Create

Synopsis: Instantiate a new instance.

Declaration:
```
constructor Create;  Virtual
```

Visibility: public

Description: `Create` does nothing in `TTextReader`.

See also: ReadLine (1594)

63.14.5 TTextReader.Reset

Synopsis: Reset the reader to the start position.

Declaration:
```
procedure Reset;  Virtual;  Abstract
```

Visibility: public

Description: `Reset` resets the position to the start of the text data.

This is an abstract call which must be implemented by descendents.

See also: TTextReader.Close (1594), TTextReader.ReadLine (1594)

63.14.6 TTextReader.Close

Synopsis: Close the text data stream.

Declaration:
```
procedure Close;  Virtual;  Abstract
```

Visibility: public

Description: `Close` closes the data stream. No ReadLine (1594) call can be performed after a call to `Close`.

See also: TTextReader.Reset (1594), TTextReader.ReadLine (1594)

63.14.7 TTextReader.ReadLine

Synopsis: Read a line of text.

Declaration:
```
procedure ReadLine(out AString: AnsiString);  Virtual;  Abstract
                  ;  Overload
function ReadLine : AnsiString;  Overload
```

Visibility: public

Description: `ReadLine` will read a line of text from the text data source. A line of text is delimited by a CRLF character pair, a LF character or a CR character. The line ending characters are not included in the string.

The method exists in 2 versions: one function where the line of text is returned as the function result, one procedure where the line of text is returned in the `AString` parameter.

See also: Eof (1595)

63.14.8 TTextReader.Eof

Synopsis: Check whether the end of the text data is returned.

Declaration:
```
Property Eof : Boolean
```

Visibility: public

Access: Read

Description: `Eof` is `True` if no more data is available for reading. If there is still data, then it is `False`.

See also: TTextReader.ReadLine (1594)

63.14.9 TTextReader.EndOfStream

Declaration:
```
Property EndOfStream : Boolean
```

Visibility: public

Access: Read

63.15 TTextWriter

63.15.1 Method overview

Page	Method	Description
1595	Close	
1595	Flush	
1595	Write	
1596	WriteLine	

63.15.2 TTextWriter.Close

Declaration:
```
procedure Close;  Virtual;  Abstract
```

Visibility: public

63.15.3 TTextWriter.Flush

Declaration:
```
procedure Flush;  Virtual;  Abstract
```

Visibility: public

63.15.4 TTextWriter.Write

Declaration:
```
procedure Write(aValue: Boolean);  Virtual;  Abstract;  Overload
procedure Write(aValue: Char);  Virtual;  Abstract;  Overload
procedure Write(aValue: Char; aCount: Integer);  Virtual;  Overload
procedure Write(const aValue: TCharArray);  Virtual;  Abstract
              ;  Overload
procedure Write(aValue: Double);  Virtual;  Abstract;  Overload
procedure Write(aValue: Integer);  Virtual;  Abstract;  Overload
procedure Write(aValue: Int64);  Virtual;  Abstract;  Overload
```

```
procedure Write(aValue: TObject); Virtual; Abstract; Overload
procedure Write(aValue: Single); Virtual; Abstract; Overload
procedure Write(const aValue: string); Virtual; Abstract; Overload
procedure Write(aValue: Cardinal); Virtual; Abstract; Overload
procedure Write(aValue: UInt64); Virtual; Abstract; Overload
procedure Write(const Fmt: string; aArgs: Array of const); Virtual
              ; Abstract; Overload
procedure Write(const aValue: TCharArray; aIndex: Integer;
              aCount: Integer); Virtual; Abstract; Overload
```

Visibility: public

63.15.5 TTextWriter.WriteLine

Declaration:
```
procedure WriteLine; Virtual; Abstract; Overload
procedure WriteLine(aValue: Boolean); Virtual; Abstract; Overload
procedure WriteLine(aValue: Char); Virtual; Abstract; Overload
procedure WriteLine(const aValue: TCharArray); Virtual; Abstract
                  ; Overload
procedure WriteLine(aValue: Double); Virtual; Abstract; Overload
procedure WriteLine(aValue: Integer); Virtual; Abstract; Overload
procedure WriteLine(aValue: Int64); Virtual; Abstract; Overload
procedure WriteLine(aValue: TObject); Virtual; Abstract; Overload
procedure WriteLine(aValue: Single); Virtual; Abstract; Overload
procedure WriteLine(const aValue: string); Virtual; Abstract
                  ; Overload
procedure WriteLine(aValue: Cardinal); Virtual; Abstract; Overload
procedure WriteLine(aValue: UInt64); Virtual; Abstract; Overload
procedure WriteLine(const Format: string; Args: Array of const)
                  ; Virtual; Abstract; Overload
procedure WriteLine(const aValue: TCharArray; Index: Integer;
                  Count: Integer); Virtual; Abstract; Overload
```

Visibility: public

63.16 TWindowedStream

63.16.1 Description

`TWindowedStream` is a TStream (1574) descendent that can be used to provide a window on the data of another stream. The position and size of the window can be determined in the constructor, and the stream will behave as a normal stream. The actual reading (or writing) will happen on the source stream.

If the source stream has some limitations (e.g. no Seek (**??**)) then the `TWindowedStream` will inherit these limitations.

The `TWindowedStream` will keep track of the last position it used, and will attempt to restore it if it was modified between calls to `Read` and `Write`.

See also: TStream (1574), TWindowedStream.Create (1597)

63.16.2 Method overview

Page	Method	Description
1597	Create	Initialize a new instance of TWindowedStream.
1597	Destroy	Destroy the TWindowedStream instance.
1597	Read	Read data from the stream.
1598	Seek	Reposition the stream.
1598	Write	Read data to the stream.

63.16.3 TWindowedStream.Create

Synopsis: Initialize a new instance of TWindowedStream.

Declaration:
```
constructor Create(aStream: TStream; const aSize: Int64;
                   const aPositionHere: Int64)
```

Visibility: public

Description: Create will create a new instance of TWindowedStream. The source stream aStream must be specified, as well as the start aPositionHere position of the window in the source stream and the size of the window aSize.

No checks on the validity of aPositionHere and aSize are done.

63.16.4 TWindowedStream.Destroy

Synopsis: Destroy the TWindowedStream instance.

Declaration:
```
destructor Destroy;  Override
```

Visibility: public

Description: Destroy simply calls the inherited destroy, it removes the TWindowedStream instance from memory. The source stream is not freed.

See also: TWindowedStream.Create (1597)

63.16.5 TWindowedStream.Read

Synopsis: Read data from the stream.

Declaration:
```
function Read(var aBuffer; aCount: LongInt) : LongInt;  Override
```

Visibility: public

Description: Read attempts to read data from the stream. It will attempt to restore the source stream position if it was changed since the last read, write or seek operation. It then attempts to read ACount bytes from the source stream into ABuffer and returns the number of actually read bytes. TWindowedStream.Read will only read as much data as the window allows, even if the source stream has more data available.

See also: TWindowedStream.Write (1598), TWindowedStream.Seek (1598)

63.16.6 TWindowedStream.Write

Synopsis: Read data to the stream.

Declaration:
```
function Write(const aBuffer; aCount: LongInt) : LongInt;  Override
```

Visibility: public

Description: `Write` attempts to write data to the stream. It will attempt to restore the source stream position if it was changed since the last read, write or seek operation. It then attempts to write `ACount` bytes from `ABuffer` to the source stream and returns the number of actually written bytes.

Errors: `TWindowedStream.Write` will raise an `EWriteError` exception if an attempt is made to write more bytes than will fit in the window, even if the source stream has more room available.

See also: TWindowedStream.Read (1597), TWindowedStream.Seek (1598)

63.16.7 TWindowedStream.Seek

Synopsis: Reposition the stream.

Declaration:
```
function Seek(const aOffset: Int64; aOrigin: TSeekorigin) : Int64
                  ;  Override
```

Visibility: public

Description: `Seek` will reposition the windowed stream based on `aOffset` and `aOrigin`. It will interpret `AOrigin` and `aOffset` relative to the position and size of the window, and will then call `Seek` on the source stream. It will return the new position in the windowed stream.

Errors: If the source stream does not support seek operations, an exception may be raises. If the combination of `AOrigin` and `aOffset` falls outside the valid window of the stream, an `EReadError` exception is raised.

See also: TWindowedStream.Read (1597), TWindowedStream.Write (1598)

Chapter 64

Reference for unit 'StreamIO'

64.1 Used units

Table 64.1: Used units by unit 'StreamIO'

Name	Page
Classes	**??**
System	**??**
sysutils	**??**

64.2 Overview

The StreamIO unit implements a call to reroute the input or output of a text file to a descendents of TStream (**??**).

This allows to use the standard pascal Read (**??**) and Write (**??**) functions (with all their possibilities), on streams.

64.3 Procedures and functions

64.3.1 AssignStream

Synopsis: Assign a text file to a stream.

Declaration: `procedure AssignStream(var F: Textfile; Stream: TStream)`

Visibility: default

Description: `AssignStream` assigns the stream `Stream` to file `F`. The file can subsequently be used to write to the stream, using the standard Write (**??**) calls.

Before writing, call Rewrite (**??**) on the stream. Before reading, call Reset (**??**).

Errors: if `Stream` is `Nil`, an exception will be raised.

See also: TStream (**??**), GetStream (1600)

64.3.2 GetStream

Synopsis: Return the stream, associated with a file.

Declaration: `function GetStream(var F: TTextRec) : TStream`

Visibility: default

Description: `GetStream` returns the instance of the stream that was associated with the file `F` using AssignStream (1599).

Errors: An invalid class reference will be returned if the file was not associated with a stream.

See also: AssignStream (1599), TStream (**??**)

Chapter 65

Reference for unit 'syncobjs'

65.1 Used units

Table 65.1: Used units by unit 'syncobjs'

Name	Page
System	??
system.timespan	??
sysutils	??

65.2 Overview

The syncobjs unit implements some classes which can be used when synchronizing threads in routines or classes that are used in multiple threads at once. The TCriticalSection (1603) class is a wrapper around low-level critical section routines (semaphores or mutexes). The TEventObject (1605) class can be used to send messages between threads (also known as conditional variables in POSIX threads).

65.3 Constants, types and variables

65.3.1 Constants

```
INFINITE = Cardinal(- 1)
```

Constant denoting an infinite timeout.

65.3.2 Types

```
PSecurityAttributes = Pointer
```

`PSecurityAttributes` is a dummy type used in non-windows implementations, so the calls remain Delphi compatible.

```
TEvent = TEventObject
```

TEvent is a simple alias for the TEventObject (1605) class.

```
TEventHandle = Pointer
```

TEventHandle is an opaque type and should not be used in user code.

```
THandleObjectArray = Array of THandleObject
```

```
TWaitResult = (wrSignaled,wrTimeout,wrAbandoned,wrError)
```

Table 65.2: Enumeration values for type TWaitResult

Value	Explanation
wrAbandoned	Wait operation was abandoned.
wrError	An error occurred during the wait operation.
wrSignaled	Event was signaled (triggered).
wrTimeout	Time-out period expired.

TWaitResult is used to report the result of a wait operation.

65.4 ELockException

65.4.1 Description

ELockException is provided for Delphi compatibility. It is not used in FPC.

See also: ESyncObjectException (1602), ELockRecursionException (1602)

65.5 ELockRecursionException

65.5.1 Description

ELockRecursionException is provided for Delphi compatibility. It is not used in FPC.

See also: ESyncObjectException (1602), ELockException (1602)

65.6 ESyncObjectException

65.6.1 Description

ESyncObjectException is used in the constructor of TEventObject (1605) to indicate failure to create a basic event.

See also: TEventObject (1605), ELockRecursionException (1602), ELockException (1602)

65.7 TCriticalSection

65.7.1 Description

`TCriticalSection` is a class wrapper around the low-level `TRTLCriticalSection` routines. It simply calls the RTL routines in the system unit for critical section support.

A critical section is a resource which can be owned by only 1 caller: it can be used to make sure that in a multithreaded application only 1 thread enters pieces of code protected by the critical section.

Typical usage is to protect a piece of code with the following code (`MySection` is a `TCriticalSection` instance):

```
// Previous code
  MySection.Acquire;
  Try
    // Protected code
  Finally
    MySection.Release;
  end;
  // Other code.
```

The protected code can be executed by only 1 thread at a time. This is useful for instance for list operations in multithreaded environments.

See also: Acquire (1603), Release (1603)

65.7.2 Method overview

Page	Method	Description
1603	Acquire	Enter the critical section.
1605	Create	Create a new critical section.
1605	Destroy	Destroy the criticalsection instance.
1604	Enter	Alias for `Acquire`.
1604	Leave	Alias for `Release`.
1603	Release	Leave the critical section.
1604	TryEnter	Try and obtain the critical section.

65.7.3 TCriticalSection.Acquire

Synopsis: Enter the critical section.

Declaration: `procedure Acquire;  Override`

Visibility: public

Description: `Acquire` attempts to enter the critical section. It will suspend the calling thread if the critical section is in use by another thread, and will resume as soon as the other thread has released the critical section.

See also: Release (1603)

65.7.4 TCriticalSection.Release

Synopsis: Leave the critical section.

Declaration: `procedure Release;  Override`

Visibility: public

Description: `Release` leaves the critical section. It will free the critical section so another thread waiting to enter the critical section will be awakened, and will enter the critical section. This call always returns immediately.

See also: Acquire (1603)

65.7.5 TCriticalSection.Enter

Synopsis: Alias for `Acquire`.

Declaration: `procedure Enter`

Visibility: public

Description: `Enter` just calls Acquire (1603).

See also: Leave (1604), Acquire (1603)

65.7.6 TCriticalSection.TryEnter

Synopsis: Try and obtain the critical section.

Declaration: `function TryEnter : Boolean`

Visibility: public

Description: `TryEnter` tries to enter the critical section: it returns at once and does not wait if the critical section is owned by another thread; if the current thread owns the critical section or the critical section was obtained successfully, `true` is returned. If the critical section is currently owned by another thread, `False` is returned.

Errors: None.

See also: TCriticalSection.Enter (1604)

65.7.7 TCriticalSection.Leave

Synopsis: Alias for `Release`.

Declaration: `procedure Leave`

Visibility: public

Description: `Leave` just calls Release (1603)

See also: Release (1603), Enter (1604)

65.7.8 TCriticalSection.Create

Synopsis: Create a new critical section.

Declaration: `constructor Create`

Visibility: public

Description: `Create` initializes a new critical section, and initializes the system objects for the critical section. It should be created only once for all threads, all threads should use the same critical section instance.

See also: Destroy (1605)

65.7.9 TCriticalSection.Destroy

Synopsis: Destroy the criticalsection instance.

Declaration: `destructor Destroy;  Override`

Visibility: public

Description: `Destroy` releases the system critical section resources, and removes the `TCriticalSection` instance from memory.

Errors: Any threads trying to enter the critical section when it is destroyed, will start running with an error (an exception should be raised).

See also: Create (1605), Acquire (1603)

65.8 TEventObject

65.8.1 Description

`TEventObject` encapsulates the `BasicEvent` implementation of the system unit in a class. The event can be used to notify other threads of a change in conditions. (in POSIX terms, this is a conditional variable). A thread that wishes to notify other threads creates an instance of `TEventObject` with a certain name, and posts events to it. Other threads that wish to be notified of these events should create their own instances of `TEventObject` with the same name, and wait for events to arrive.

See also: TCriticalSection (1603)

65.8.2 Method overview

Page	Method	Description
1606	Create	Create a new event object.
1606	ResetEvent	Reset the event.
1606	SetEvent	Set the event.

65.8.3 Property overview

Page	Properties	Access	Description
1606	ManualReset	r	Should the event be reset manually.

65.8.4 TEventObject.Create

Synopsis: Create a new event object.

Declaration:
```
constructor Create(EventAttributes: PSecurityAttributes;
                   AManualReset: Boolean; InitialState: Boolean;
                   const Name: string; UseComWait: Boolean);  Overload
```

Visibility: public

Description: `Create` creates a new event object with unique name `Name` (ignored on non-windows platforms). The object will be created with security attributes `EventAttributes` (this parameters is used on Windows only).

The `AManualReset` indicates whether the event must be reset manually (if it is `False`, the event is reset immediately after the first thread waiting for it is notified). `InitialState` determines whether the event is initially set or not.

See also: ManualReset (1606), ResetEvent (1606)

65.8.5 TEventObject.ResetEvent

Synopsis: Reset the event.

Declaration:
```
procedure ResetEvent
```

Visibility: public

Description: `ResetEvent` turns off the event. Any WaitFor (1605) operation will suspend the calling thread.

See also: SetEvent (1606), WaitFor (1605)

65.8.6 TEventObject.SetEvent

Synopsis: Set the event.

Declaration:
```
procedure SetEvent
```

Visibility: public

Description: `SetEvent` sets the event. If the ManualReset (1606) is `True` any thread that was waiting for the event to be set (using WaitFor (1605)) will resume it's operation. After the event was set, any thread that executes `WaitFor` will return at once. If `ManualReset` is `False`, only one thread will be notified that the event was set, and the event will be immediately reset after that.

See also: WaitFor (1605), ManualReset (1606)

65.8.7 TEventObject.ManualReset

Synopsis: Should the event be reset manually.

Declaration:
```
Property ManualReset : Boolean
```

Visibility: public

Access: Read

Description: `ManualReset` indicates whether the event must be reset manually: if it is `False`, the event is reset immediately after the first thread waiting for it is notified. if it is `True`, then the event is never reset automatically, and ResetEvent (1606) must be called manually after a thread was notified.

See also: ResetEvent (1606)

65.9 THandleObject

65.9.1 Description

`THandleObject` is an abstract parent class for synchronization classes that need to store an operating system handle. It introduces a property Handle (1607) which can be used to store the operating system handle. The handle is in no way manipulated by `THandleObject`, only storage is provided.

Do not create an instance of `THandleObject`. It is an abstract class. Recent versions of FPC actually declare the class as abstract.

See also: Handle (1607)

65.9.2 Method overview

Page	Method	Description
1607	Create	
1607	Destroy	Free the instance.
1607	WaitFor	

65.9.3 Property overview

Page	Properties	Access	Description
1607	Handle	r	Handle for this object.
1608	LastError	r	Last operating system error.

65.9.4 THandleObject.Create

Declaration: `constructor Create(UseComWait: Boolean)`

Visibility: public

65.9.5 THandleObject.Destroy

Synopsis: Free the instance.

Declaration: `destructor Destroy;  Override`

Visibility: public

Description: `Destroy` does nothing in the Free Pascal implementation of `THandleObject`.

65.9.6 THandleObject.WaitFor

Declaration:
```
function WaitFor(Timeout: Cardinal) : TWaitResult;  Overload
function WaitFor(const Timeout: TTimespan) : TWaitResult;  Overload
```

Visibility: public

65.9.7 THandleObject.Handle

Synopsis: Handle for this object.

Declaration: `Property Handle : TEventHandle`

Visibility: public

Access: Read

Description: `Handle` provides read-only access to the operating system handle of this instance. The public access is read-only, descendent classes should set the handle by accessing it's protected field `FHandle` directly.

65.9.8 THandleObject.LastError

Synopsis: Last operating system error.

Declaration: `Property LastError : Integer`

Visibility: public

Access: Read

Description: `LastError` provides read-only access to the last operating system error code for operations on Handle (1607).

See also: Handle (1607)

65.10 TSimpleEvent

65.10.1 Description

`TSimpleEvent` is a simple descendent of the TEventObject (1605) class. It creates an event with no name, which must be reset manually, and which is initially not set.

See also: TEventObject (1605), TSimpleEvent.Create (1608)

65.10.2 Method overview

Page	Method	Description
1608	Create	Creates a new `TSimpleEvent` instance.

65.10.3 TSimpleEvent.Create

Synopsis: Creates a new `TSimpleEvent` instance.

Declaration: `constructor Create`

Visibility: default

Description: `Create` instantiates a new `TSimpleEvent` instance. It simply calls the inherited Create (1606) with `Nil` for the security attributes, an empty name, `AManualReset` set to `True`, and `InitialState` to `False`.

See also: TEventObject.Create (1606)

65.11 TSynchroObject

65.11.1 Description

`TSynchroObject` is an abstract synchronization resource object. It implements 2 virtual methods Acquire (1609) which can be used to acquire the resource, and Release (1609) to release the resource.

See also: Acquire (1609), Release (1609)

65.11.2 Method overview

Page	Method	Description
1609	Acquire	Acquire synchronization resource.
1609	Release	Release previously acquired synchronization resource.

65.11.3 TSynchroObject.Acquire

Synopsis: Acquire synchronization resource.

Declaration: `procedure Acquire; Virtual`

Visibility: default

Description: `Acquire` does nothing in `TSynchroObject`. Descendent classes must override this method to acquire the resource they manage.

See also: Release (1609)

65.11.4 TSynchroObject.Release

Synopsis: Release previously acquired synchronization resource.

Declaration: `procedure Release; Virtual`

Visibility: default

Description: `Release` does nothing in `TSynchroObject`. Descendent classes must override this method to release the resource they acquired through the Acquire (1609) call.

See also: Acquire (1609)

Chapter 66

Reference for unit 'URIParser'

66.1 Used units

Table 66.1: Used units by unit 'URIParser'

Name	Page
System	??

66.2 Overview

The URIParser unit contains a basic type (TURI (1616)) and some routines for the parsing (ParseURI (1614)) and construction (EncodeURI (1613)) of Uniform Resource Indicators, commonly referred to as URL: Uniform Resource Location. It is used in various other units, and in itself contains no classes. It supports all protocols, username/password/port specification, query parameters and bookmarks etc..

66.3 Basic usage of URI parsing functions

The `URIParser` unit provides simple functions to parse and construct URIs. The main function is `ParseURI` which parses a URI string into a `TURI` record containing all the components.

The following example demonstrates basic URI parsing and reconstruction:

```
program BasicURIExample;

{$mode objfpc}{$h+}

uses
  URIParser, SysUtils;

procedure ParseAndDisplay(const URIString: string);
var
  URI: TURI;
begin
```

```
  WriteLn('Original URI: ', URIString);

  // Parse the URI
  URI := ParseURI(URIString);

  // Display all components
  WriteLn('  Protocol: ', URI.Protocol);
  WriteLn('  Username: ', URI.Username);
  WriteLn('  Password: ', URI.Password);
  WriteLn('  Host: ', URI.Host);
  WriteLn('  Port: ', URI.Port);
  WriteLn('  Path: ', URI.Path);
  WriteLn('  Document: ', URI.Document);
  WriteLn('  Params: ', URI.Params);
  WriteLn('  Bookmark: ', URI.Bookmark);
  WriteLn('  Has Authority: ', URI.HasAuthority);

  // Reconstruct the URI
  WriteLn('  Reconstructed: ', EncodeURI(URI));
  WriteLn;
end;

begin
  // Parse different types of URIs
  ParseAndDisplay('https://user:pass@www.example.com:8080/path/to/page.html?param1=v
  ParseAndDisplay('ftp://files.example.org/pub/software/myfile.tar.gz');
  ParseAndDisplay('mailto:user@example.com');
  ParseAndDisplay('file:///home/user/document.txt');
  ParseAndDisplay('/relative/path/file.html');
end.
```

The following example shows how to work with relative URIs and file paths:

```
program URIUtilitiesExample;

{$mode objfpc}{$h+}

uses
  URIParser, SysUtils;

var
  BaseURI, RelativeURI, AbsoluteURI: string;
  Filename: string;
  URI: string;

begin
  WriteLn('=== Relative URI Resolution ===');
  BaseURI := 'https://www.example.com/docs/manual/';
  RelativeURI := '../images/logo.png';

  if ResolveRelativeURI(BaseURI, RelativeURI, AbsoluteURI) then
  begin
    WriteLn('Base URI: ', BaseURI);
    WriteLn('Relative URI: ', RelativeURI);
```

```
    WriteLn('Absolute URI: ', AbsoluteURI);
  end
  else
    WriteLn('Failed to resolve relative URI');

  WriteLn;
  WriteLn('=== File/URI Conversion ===');

  // Convert filename to URI
  Filename := '/home/user/documents/readme.txt';
  URI := FilenameToURI(Filename);
  WriteLn('Filename: ', Filename);
  WriteLn('File URI: ', URI);

  // Convert URI back to filename
  if URIToFilename(URI, Filename) then
  begin
    WriteLn('Converted back to filename: ', Filename);
  end
  else
    WriteLn('Failed to convert URI to filename');

  WriteLn;
  WriteLn('=== URI Type Checking ===');

  // Check if URIs are absolute
  URI := 'https://www.example.com/page.html';
  WriteLn('URI: ', URI, ' - Is absolute: ', IsAbsoluteURI(URI));

  URI := '/relative/path.html';
  WriteLn('URI: ', URI, ' - Is absolute: ', IsAbsoluteURI(URI));

  URI := 'mailto:test@example.com';
  WriteLn('URI: ', URI, ' - Is absolute: ', IsAbsoluteURI(URI));
end.
```

The following example demonstrates URI parsing with default values:

```
program DefaultsURIExample;

{$mode objfpc}{$h+}

uses
  URIParser, SysUtils;

procedure ParseWithDefaults(const URIString: string; const DefaultProtocol: string; I
var
  URI: TURI;
begin
  WriteLn('Parsing: ', URIString);
  WriteLn('Default protocol: ', DefaultProtocol);
  WriteLn('Default port: ', DefaultPort);

  // Parse with default protocol and port
```

```
  URI := ParseURI(URIString, DefaultProtocol, DefaultPort);

  WriteLn('Results:');
  WriteLn('  Protocol: ', URI.Protocol);
  WriteLn('  Host: ', URI.Host);
  WriteLn('  Port: ', URI.Port);
  WriteLn('  Path: ', URI.Path);
  WriteLn('  Document: ', URI.Document);
  WriteLn('  Full URI: ', EncodeURI(URI));
  WriteLn;
end;

begin
  // Parse URIs with missing protocol/port using defaults
  ParseWithDefaults('www.example.com/index.html', 'http', 80);
  ParseWithDefaults('ftp.example.org/pub/files/', 'ftp', 21);
  ParseWithDefaults('mail.example.com:993/inbox', 'imaps', 993);

  // Parse complete URIs (defaults are ignored)
  ParseWithDefaults('https://secure.example.com:443/login', 'http', 80);
end.
```

Key functions in the `URIParser` unit:

- `ParseURI` - Parses a URI string into components
- `EncodeURI` - Reconstructs a URI string from components
- `ResolveRelativeURI` - Resolves relative URIs against a base URI
- `URIToFilename` - Converts file URIs to local filenames
- `FilenameToURI` - Converts local filenames to file URIs
- `IsAbsoluteURI` - Checks if a URI is absolute (has a protocol)

See also: ParseURI (1614), EncodeURI (1613), TURI (1616), ResolveRelativeURI (1614)

66.4 Constants, types and variables

66.4.1 Types

66.5 Procedures and functions

66.5.1 EncodeURI

Synopsis: Form a string representation of the URI.

Declaration: `function EncodeURI(const URI: TURI) : string`

Visibility: default

Description: `EncodeURI` will return a valid text representation of the URI in the `URI` record.

See also: ParseURI (1614)

66.5.2 FilenameToURI

Synopsis: Construct a URI from a filename.

Declaration:
```
function FilenameToURI(const Filename: string; Encode: Boolean) : string
```

Visibility: default

Description: `FilenameToURI` takes `Filename` and constructs a `file:` protocol URI from it.

Errors: None.

See also: URIToFilename (1615)

66.5.3 IsAbsoluteURI

Synopsis: Check whether a URI is absolute.

Declaration:
```
function IsAbsoluteURI(const UriReference: string) : Boolean
```

Visibility: default

Description: `IsAbsoluteURI` returns `True` if the URI in `UriReference` is absolute, i.e. contains a protocol part.

Errors: None.

See also: FilenameToURI (1614), URIToFileName (1615)

66.5.4 ParseURI

Synopsis: Parse a URI and split it into its constituent parts.

Declaration:
```
function ParseURI(const URI: string; Decode: Boolean) : TURI;  Overload
function ParseURI(const URI: string; const DefaultProtocol: string;
                  DefaultPort: Word; Decode: Boolean) : TURI;  Overload
```

Visibility: default

Description: `ParseURI` decodes `URI` and returns the various parts of the URI in the result record.

The function accepts the most general URI scheme:

```
proto://user:pwd@host:port/path/document?params#bookmark
```

Missing (optional) parts in the URI will be left blank in the result record. If a default protocol and port are specified, they will be used in the record if the corresponding part is not present in the URI.

See also: EncodeURI (1613)

66.5.5 ResolveRelativeURI

Synopsis: Return a relative link.

Declaration:
```
function ResolveRelativeURI(const BaseUri: UnicodeString;
                            const RelUri: UnicodeString;
                            out ResultUri: UnicodeString) : Boolean
                            ;  Overload
function ResolveRelativeURI(const BaseUri: AnsiString;
                            const RelUri: AnsiString;
                            out ResultUri: AnsiString) : Boolean
                            ;  Overload
```

Visibility: default

Description: `ResolveRelativeURI` returns in `ResultUri` an absolute link constructed from a base URI `BaseURI` and a relative link `RelURI`. One of the two URI names must have a protocol specified. If the `RelURI` argument contains a protocol, it is considered a complete (absolute) URI and is returned as the result.

The function returns `True` if a link was successfully returned.

Errors: If no protocols are specified, the function returns `False`

66.5.6 URIToFilename

Synopsis: Convert a URI to a filename.

Declaration:
```
function URIToFilename(const URI: string; out Filename: string)
                       : Boolean
```

Visibility: default

Description: `URIToFilename` returns a filename (using the correct Path Delimiter character) from `URI`. The `URI` must be of protocol `File` or have no protocol.

Errors: If the URI contains an unsupported protocol, `False` is returned.

See also: ResolveRelativeURI (1614), FilenameToURI (1614)

66.6 TURI

```
TURI = record
public
  Protocol : string;
  Username : string;
  Password
  : string;
  Host : string;
  Port : Word;
  Path : string;
  Document
  : string;
  Params : string;
  Bookmark : string;
  HasAuthority
  : Boolean;
  create;
  function ToString : string;
```

```
  function URI
  : string;
  procedure Free;
end
```

TURI is the basic record that can be filled by the ParseURI (1614) call. It contains the contents of a URI, parsed out in it's various pieces.

66.6.1 Method overview

Page	Method	Description
1616	create	
1616	Free	
1616	ToString	
1616	URI	

66.6.2 TURI.create

Declaration:
```
constructor create(aURI: string; aDecode: Boolean)
constructor create(const aURI: string; const aDefaultProtocol: string;
                   aDefaultPort: Word; aDecode: Boolean)
```

Visibility: public

66.6.3 TURI.ToString

Declaration:
```
function ToString : string
```

Visibility: public

66.6.4 TURI.URI

Declaration:
```
function URI : string
```

Visibility: public

66.6.5 TURI.Free

Declaration:
```
procedure Free
```

Visibility: public

Chapter 67

Reference for unit 'wformat'

67.1 Used units

Table 67.1: Used units by unit 'wformat'

Name	Page
Classes	??
System	??
sysutils	??

67.2 Overview

The `wformat` unit provides the abstract base class TFormattingWriter (1618) for creating formatted document writers. This class defines a standard interface for generating formatted output in various formats like HTML, LaTeX, or plain text. Descendant classes implement the formatting methods for their specific output format.

67.3 Constants, types and variables

67.3.1 Constants

```
LineFeed = LineEnding
```

`LineFeed` is a constant that contains the platform-specific line ending character sequence. It is defined as `LineEnding` and is used by DumpLn (1620) for consistent line breaks across platforms.

67.3.2 Types

```
TlistType = (ltNumbered,ltOrdered,ltDefinition)
```

Table 67.2: Enumeration values for type TlistType

Value	Explanation
ltDefinition	Definition list with name-value pairs
ltNumbered	Numbered list with numeric indices
ltOrdered	Ordered list with bullets or markers

`TListType` defines the types of lists that can be created using the TFormattingWriter (1618) class.

67.4 TFormattingWriter

67.4.1 Description

`TFormattingWriter` is an abstract base class that defines the interface for creating formatted documents. It provides methods for document structure (headers, paragraphs), text formatting (bold, italic, underline), tables, and lists. Descendant classes implement the specific formatting for different output formats such as HTML, LaTeX, or plain text.

The class writes output to a stream provided during construction. Most methods are virtual and should be overridden by descendant classes to provide format-specific implementation.

See also: TListType (1617)

67.4.2 Method overview

Page	Method	Description
1623	BoldEnd	Ends bold text formatting
1623	BoldStart	Begins bold text formatting
1626	CellEnd	Ends the current table cell
1626	CellNext	Moves to the next table cell
1626	CellStart	Begins a new table cell
1619	Create	Creates a new formatting writer instance
1629	DefinitionItem	Creates a definition list item with name and description
1621	DocumentEnd	Ends the current document
1621	DocumentStart	Begins a new document
1620	Dump	Writes raw text directly to the output stream
1620	DumpLn	Writes raw text followed by line ending to the output stream
1620	EscapeText	Escapes text for the target format
1621	Header	Writes a complete header at the specified level
1627	HeaderCellEnd	Ends the current table header cell
1627	HeaderCellNext	Moves to the next table header cell
1627	HeaderCellStart	Begins a new table header cell
1622	HeaderEnd	Ends a header at the specified level
1622	HeaderStart	Begins a header at the specified level
1623	ItalicEnd	Ends italic text formatting
1623	ItalicStart	Begins italic text formatting
1622	LineBreak	Inserts a line break
1628	ListEnd	Ends a list of the specified type
1628	ListItem	Creates a complete list item with text
1628	ListItemEnd	Ends the current list item
1628	ListItemStart	Begins a new list item
1627	ListStart	Begins a list of the specified type
1622	ParagraphEnd	Ends the current paragraph
1622	ParagraphStart	Begins a new paragraph
1624	PreformatEnd	Ends preformatted text block
1624	PreformatStart	Begins preformatted text block
1625	RowEnd	Ends the current table row
1626	RowNext	Moves to the next table row
1625	RowStart	Begins a new table row
1623	Rule	Inserts a horizontal rule
1625	TableEnd	Ends the current table
1625	TableStart	Begins a table with specified columns and border
1624	UnderlineEnd	Ends underlined text formatting
1624	UnderlineStart	Begins underlined text formatting
1620	Write	Writes escaped text to the output stream
1621	WriteFmt	Writes formatted text using Format function
1629	WriteList	Writes a complete list from string list

67.4.3 TFormattingWriter.Create

Synopsis: Creates a new formatting writer instance

Declaration:
```
constructor Create(AStream: TStream);  Virtual
```

Visibility: public

Description: Creates a new `TFormattingWriter` instance that will write output to the specified stream. The stream must be valid and remain available for the lifetime of the writer instance.

67.4.4 TFormattingWriter.EscapeText

Synopsis: Escapes text for the target format

Declaration: `function EscapeText(AText: string) : string;  Virtual`

Visibility: public

Description: Virtual method that escapes special characters in text for the target output format. The base implementation returns the text unchanged. Descendant classes should override this method to escape format-specific characters.

See also: TFormattingWriter.Write (1620)

67.4.5 TFormattingWriter.Dump

Synopsis: Writes raw text directly to the output stream

Declaration: `procedure Dump(const AText: string)`

Visibility: public

Description: Writes text directly to the output stream without any escaping or formatting. This is used for writing format-specific markup or pre-escaped text.

See also: TFormattingWriter.DumpLn (1620), TFormattingWriter.Write (1620)

67.4.6 TFormattingWriter.DumpLn

Synopsis: Writes raw text followed by line ending to the output stream

Declaration: `procedure DumpLn(const AText: string)`

Visibility: public

Description: Writes text directly to the output stream followed by a line ending without any escaping or formatting. This is used for writing format-specific markup or pre-escaped text that should end with a new line.

See also: TFormattingWriter.Dump (1620), LineFeed (1617)

67.4.7 TFormattingWriter.Write

Synopsis: Writes escaped text to the output stream

Declaration: `procedure Write(const AText: string)`

Visibility: public

Description: Writes text to the output stream after passing it through EscapeText (1620) for proper formatting. This is the preferred method for writing user content that needs format-specific escaping.

See also: TFormattingWriter.EscapeText (1620), TFormattingWriter.Dump (1620), TFormattingWriter.WriteFmt (1621)

67.4.8 TFormattingWriter.WriteFmt

Synopsis: Writes formatted text using Format function

Declaration: `procedure WriteFmt(const Fmt: string; Args: Array of const)`

Visibility: public

Description: Formats the text using the Format function and writes the result through Write (1620). This provides a convenient way to write formatted text with proper escaping.

See also: TFormattingWriter.Write (1620)

67.4.9 TFormattingWriter.DocumentStart

Synopsis: Begins a new document

Declaration: `procedure DocumentStart(const Title: string);  Virtual`

Visibility: public

Description: Virtual method that begins a new document with the specified title. Descendant classes should override this method to write format-specific document headers or initialization code.

See also: TFormattingWriter.DocumentEnd (1621)

67.4.10 TFormattingWriter.DocumentEnd

Synopsis: Ends the current document

Declaration: `procedure DocumentEnd;  Virtual`

Visibility: public

Description: Virtual method that ends the current document. Descendant classes should override this method to write format-specific document footers or cleanup code.

See also: TFormattingWriter.DocumentStart (1621)

67.4.11 TFormattingWriter.Header

Synopsis: Writes a complete header at the specified level

Declaration: `procedure Header(Alevel: Integer; Msg: string)`

Visibility: public

Description: Writes a complete header by calling HeaderStart (1622), writing the message text, and calling HeaderEnd (1622). This is a convenience method for writing simple headers.

See also: TFormattingWriter.HeaderStart (1622), TFormattingWriter.HeaderEnd (1622)

67.4.12 TFormattingWriter.HeaderStart

Synopsis: Begins a header at the specified level

Declaration: `procedure HeaderStart(Alevel: Integer);  Virtual`

Visibility: public

Description: Virtual method that begins header formatting at the specified level. Descendant classes should override this method to write format-specific header opening tags or formatting.

See also: TFormattingWriter.HeaderEnd (1622), TFormattingWriter.Header (1621)

67.4.13 TFormattingWriter.HeaderEnd

Synopsis: Ends a header at the specified level

Declaration: `procedure HeaderEnd(Alevel: Integer);  Virtual`

Visibility: public

Description: Virtual method that ends header formatting at the specified level. Descendant classes should override this method to write format-specific header closing tags or formatting.

See also: TFormattingWriter.HeaderStart (1622), TFormattingWriter.Header (1621)

67.4.14 TFormattingWriter.ParagraphStart

Synopsis: Begins a new paragraph

Declaration: `procedure ParagraphStart;  Virtual`

Visibility: public

Description: Virtual method that begins paragraph formatting. Descendant classes should override this method to write format-specific paragraph opening tags or formatting.

See also: TFormattingWriter.ParagraphEnd (1622)

67.4.15 TFormattingWriter.ParagraphEnd

Synopsis: Ends the current paragraph

Declaration: `procedure ParagraphEnd;  Virtual`

Visibility: public

Description: Virtual method that ends paragraph formatting. Descendant classes should override this method to write format-specific paragraph closing tags or formatting.

See also: TFormattingWriter.ParagraphStart (1622)

67.4.16 TFormattingWriter.LineBreak

Synopsis: Inserts a line break

Declaration: `procedure LineBreak;  Virtual`

Visibility: public

Description: Virtual method that inserts a line break in the output. Descendant classes should override this method to write format-specific line break markup.

67.4.17 TFormattingWriter.Rule

Synopsis: Inserts a horizontal rule

Declaration: `procedure Rule;  Virtual`

Visibility: public

Description: Virtual method that inserts a horizontal rule or separator line in the output. Descendant classes should override this method to write format-specific rule markup.

67.4.18 TFormattingWriter.BoldStart

Synopsis: Begins bold text formatting

Declaration: `procedure BoldStart;  Virtual`

Visibility: public

Description: Virtual method that begins bold text formatting. Descendant classes should override this method to write format-specific bold opening markup.

See also: TFormattingWriter.BoldEnd (1623)

67.4.19 TFormattingWriter.BoldEnd

Synopsis: Ends bold text formatting

Declaration: `procedure BoldEnd;  Virtual`

Visibility: public

Description: Virtual method that ends bold text formatting. Descendant classes should override this method to write format-specific bold closing markup.

See also: TFormattingWriter.BoldStart (1623)

67.4.20 TFormattingWriter.ItalicStart

Synopsis: Begins italic text formatting

Declaration: `procedure ItalicStart;  Virtual`

Visibility: public

Description: Virtual method that begins italic text formatting. Descendant classes should override this method to write format-specific italic opening markup.

See also: TFormattingWriter.ItalicEnd (1623)

67.4.21 TFormattingWriter.ItalicEnd

Synopsis: Ends italic text formatting

Declaration: `procedure ItalicEnd;  Virtual`

Visibility: public

Description: Virtual method that ends italic text formatting. Descendant classes should override this method to write format-specific italic closing markup.

See also: TFormattingWriter.ItalicStart (1623)

67.4.22 TFormattingWriter.UnderlineStart

Synopsis: Begins underlined text formatting

Declaration: `procedure UnderlineStart; Virtual`

Visibility: public

Description: Virtual method that begins underlined text formatting. Descendant classes should override this method to write format-specific underline opening markup.

See also: TFormattingWriter.UnderlineEnd (1624)

67.4.23 TFormattingWriter.UnderlineEnd

Synopsis: Ends underlined text formatting

Declaration: `procedure UnderlineEnd; Virtual`

Visibility: public

Description: Virtual method that ends underlined text formatting. Descendant classes should override this method to write format-specific underline closing markup.

See also: TFormattingWriter.UnderlineStart (1624)

67.4.24 TFormattingWriter.PreformatStart

Synopsis: Begins preformatted text block

Declaration: `procedure PreformatStart; Virtual`

Visibility: public

Description: Virtual method that begins a preformatted text block where whitespace and line breaks are preserved. Descendant classes should override this method to write format-specific preformatted opening markup.

See also: TFormattingWriter.PreformatEnd (1624)

67.4.25 TFormattingWriter.PreformatEnd

Synopsis: Ends preformatted text block

Declaration: `procedure PreformatEnd; Virtual`

Visibility: public

Description: Virtual method that ends a preformatted text block. Descendant classes should override this method to write format-specific preformatted closing markup.

See also: TFormattingWriter.PreformatStart (1624)

67.4.26 TFormattingWriter.TableStart

Synopsis: Begins a table with specified columns and border

Declaration: `procedure TableStart(NoCols: Integer; Border: Boolean);  Virtual`

Visibility: public

Description: Virtual method that begins a table with the specified number of columns and optional border. Descendant classes should override this method to write format-specific table opening markup.

See also: TFormattingWriter.TableEnd (1625)

67.4.27 TFormattingWriter.TableEnd

Synopsis: Ends the current table

Declaration: `procedure TableEnd;  Virtual`

Visibility: public

Description: Virtual method that ends the current table. Descendant classes should override this method to write format-specific table closing markup.

See also: TFormattingWriter.TableStart (1625)

67.4.28 TFormattingWriter.RowStart

Synopsis: Begins a new table row

Declaration: `procedure RowStart;  Virtual`

Visibility: public

Description: Virtual method that begins a new table row. Descendant classes should override this method to write format-specific row opening markup.

See also: TFormattingWriter.RowEnd (1625), TFormattingWriter.RowNext (1626)

67.4.29 TFormattingWriter.RowEnd

Synopsis: Ends the current table row

Declaration: `procedure RowEnd;  Virtual`

Visibility: public

Description: Virtual method that ends the current table row. Descendant classes should override this method to write format-specific row closing markup.

See also: TFormattingWriter.RowStart (1625), TFormattingWriter.RowNext (1626)

67.4.30 TFormattingWriter.RowNext

Synopsis: Moves to the next table row

Declaration: `procedure RowNext`

Visibility: public

Description: Convenience method that calls RowEnd (1625) followed by RowStart (1625) to move to the next table row.

See also: TFormattingWriter.RowStart (1625), TFormattingWriter.RowEnd (1625)

67.4.31 TFormattingWriter.CellStart

Synopsis: Begins a new table cell

Declaration: `procedure CellStart;  Virtual`

Visibility: public

Description: Virtual method that begins a new table cell. Descendant classes should override this method to write format-specific cell opening markup.

See also: TFormattingWriter.CellEnd (1626), TFormattingWriter.CellNext (1626)

67.4.32 TFormattingWriter.CellEnd

Synopsis: Ends the current table cell

Declaration: `procedure CellEnd;  Virtual`

Visibility: public

Description: Virtual method that ends the current table cell. Descendant classes should override this method to write format-specific cell closing markup.

See also: TFormattingWriter.CellStart (1626), TFormattingWriter.CellNext (1626)

67.4.33 TFormattingWriter.CellNext

Synopsis: Moves to the next table cell

Declaration: `procedure CellNext`

Visibility: public

Description: Convenience method that calls CellEnd (1626) followed by CellStart (1626) to move to the next table cell.

See also: TFormattingWriter.CellStart (1626), TFormattingWriter.CellEnd (1626)

67.4.34 TFormattingWriter.HeaderCellStart

Synopsis: Begins a new table header cell

Declaration: `procedure HeaderCellStart;  Virtual`

Visibility: public

Description: Virtual method that begins a new table header cell. Descendant classes should override this method to write format-specific header cell opening markup.

See also: TFormattingWriter.HeaderCellEnd (1627), TFormattingWriter.HeaderCellNext (1627)

67.4.35 TFormattingWriter.HeaderCellEnd

Synopsis: Ends the current table header cell

Declaration: `procedure HeaderCellEnd;  Virtual`

Visibility: public

Description: Virtual method that ends the current table header cell. Descendant classes should override this method to write format-specific header cell closing markup.

See also: TFormattingWriter.HeaderCellStart (1627), TFormattingWriter.HeaderCellNext (1627)

67.4.36 TFormattingWriter.HeaderCellNext

Synopsis: Moves to the next table header cell

Declaration: `procedure HeaderCellNext`

Visibility: public

Description: Convenience method that calls HeaderCellEnd (1627) followed by HeaderCellStart (1627) to move to the next table header cell.

See also: TFormattingWriter.HeaderCellStart (1627), TFormattingWriter.HeaderCellEnd (1627)

67.4.37 TFormattingWriter.ListStart

Synopsis: Begins a list of the specified type

Declaration: `procedure ListStart(ListType: TlistType);  Virtual`

Visibility: public

Description: Virtual method that begins a list of the specified type. Descendant classes should override this method to write format-specific list opening markup.

See also: TFormattingWriter.ListEnd (1628), TListType (1617)

67.4.38 TFormattingWriter.ListEnd

Synopsis: Ends a list of the specified type

Declaration: `procedure ListEnd(ListType: TlistType);  Virtual`

Visibility: public

Description: Virtual method that ends a list of the specified type. Descendant classes should override this method to write format-specific list closing markup.

See also: TFormattingWriter.ListStart (1627), TListType (1617)

67.4.39 TFormattingWriter.ListItemStart

Synopsis: Begins a new list item

Declaration: `procedure ListItemStart;  Virtual`

Visibility: public

Description: Virtual method that begins a new list item. Descendant classes should override this method to write format-specific list item opening markup.

See also: TFormattingWriter.ListItemEnd (1628), TFormattingWriter.ListItem (1628)

67.4.40 TFormattingWriter.ListItemEnd

Synopsis: Ends the current list item

Declaration: `procedure ListItemEnd;  Virtual`

Visibility: public

Description: Virtual method that ends the current list item. Descendant classes should override this method to write format-specific list item closing markup.

See also: TFormattingWriter.ListItemStart (1628), TFormattingWriter.ListItem (1628)

67.4.41 TFormattingWriter.ListItem

Synopsis: Creates a complete list item with text

Declaration: `procedure ListItem(const AText: string)`

Visibility: public

Description: Convenience method that creates a complete list item by calling ListItemStart (1628), writing the text, and calling ListItemEnd (1628).

See also: TFormattingWriter.ListItemStart (1628), TFormattingWriter.ListItemEnd (1628)

67.4.42 TFormattingWriter.DefinitionItem

Synopsis: Creates a definition list item with name and description

Declaration:
```
procedure DefinitionItem(const Aname: string; const AText: string)
                         ;  Virtual
```

Visibility: public

Description: Virtual method that creates a definition list item with a name and its description. Descendant classes should override this method to write format-specific definition item markup.

See also: TlistType.ltDefinition (1617)

67.4.43 TFormattingWriter.WriteList

Synopsis: Writes a complete list from string list

Declaration:
```
procedure WriteList(ListType: TlistType; List: TStrings)
```

Visibility: public

Description: Convenience method that writes a complete list from a `TStrings` object. For definition lists, strings should contain name=value pairs where the name is separated from the value by an equals sign.

See also: TFormattingWriter.ListStart (1627), TFormattingWriter.ListEnd (1628), TFormattingWriter.ListItem (1628), TFormattingWriter.DefinitionItem (1629), TListType (1617)

Chapter 68

Reference for unit 'whtml'

68.1 Used units

Table 68.1: Used units by unit 'whtml'

Name	Page
Classes	??
System	??
sysutils	??
wformat	1617

68.2 Overview

The `whtml` unit provides THTMLWriter (1630), a concrete implementation of TFormattingWriter (1630) that generates HTML output. This class implements all the abstract formatting methods to produce valid HTML markup for documents, forms, tables, and lists.

68.3 Constants, types and variables

68.3.1 Constants

```
ListTags : Array[TListType] of string = ('OL', 'UL', 'DL')
```

`ListTags` is an array constant that maps TListType (1630) enumeration values to their corresponding HTML list tags. Contains 'OL' for numbered lists, 'UL' for ordered lists, and 'DL' for definition lists.

68.4 THTMLWriter

68.4.1 Description

`THTMLWriter` is a concrete implementation of TFormattingWriter (1630) that generates HTML output. It implements all formatting methods to produce valid HTML markup including document

structure, text formatting, tables, lists, and HTML forms. The class properly escapes HTML special characters and generates standard HTML tags.

See also: TFormattingWriter (1630), ListTags (1630)

68.4.2 Method overview

Page	Method	Description
1634	BoldEnd	Ends HTML bold formatting
1634	BoldStart	Begins HTML bold formatting
1637	CellEnd	Ends HTML table cell
1637	CellStart	Begins HTML table cell
1632	Create	Creates a new HTML writer instance
1638	DefinitionItem	Creates HTML definition list item
1633	DocumentEnd	Ends the HTML document
1633	DocumentStart	Begins an HTML document
1640	EmitArea	Emits HTML textarea for multi-line input
1641	EmitButton	Emits HTML button input
1640	EmitCheckBox	Emits HTML checkbox input
1640	EmitComboBox	Emits HTML select dropdown list
1641	EmitHiddenVar	Emits HTML hidden input field
1639	EmitInput	Emits HTML input field
1639	EmitPasswordInput	Emits HTML password input field
1640	EmitRadioButton	Emits HTML radio button input
1641	EmitResetButton	Emits HTML reset button
1641	EmitSubmitButton	Emits HTML submit button
1632	EscapeText	Escapes HTML special characters
1639	FormEnd	Ends HTML form
1639	FormStart	Begins HTML form with action and method
1637	HeaderCellEnd	Ends HTML table header cell
1637	HeaderCellStart	Begins HTML table header cell
1633	HeaderEnd	Ends HTML header tag at specified level
1633	HeaderStart	Begins HTML header tag at specified level
1635	ItalicEnd	Ends HTML italic formatting
1635	ItalicStart	Begins HTML italic formatting
1634	LineBreak	Inserts HTML line break tag
1638	ListEnd	Ends HTML list of specified type
1638	ListItemEnd	Ends HTML list item
1638	ListItemStart	Begins HTML list item
1637	ListStart	Begins HTML list of specified type
1634	ParagraphEnd	Ends HTML paragraph tag
1633	ParagraphStart	Begins HTML paragraph tag
1636	PreformatEnd	Ends HTML preformatted text block
1635	PreformatStart	Begins HTML preformatted text block
1636	RowEnd	Ends HTML table row
1636	RowStart	Begins HTML table row
1634	Rule	Inserts HTML horizontal rule tag
1636	TableEnd	Ends HTML table
1636	TableStart	Begins HTML table with optional border
1632	TagEnd	Writes a closing HTML tag
1632	TagStart	Writes an opening HTML tag with optional attributes
1635	UnderlineEnd	Ends HTML underline formatting
1635	UnderlineStart	Begins HTML underline formatting

68.4.3 THTMLWriter.Create

Synopsis: Creates a new HTML writer instance

Declaration: `constructor Create(AStream: TStream);  Override`

Visibility: public

Description: Creates a new `THTMLWriter` instance that will write HTML output to the specified stream. This constructor calls the inherited constructor to initialize the base formatting writer.

See also: TFormattingWriter.Create (1630)

68.4.4 THTMLWriter.TagStart

Synopsis: Writes an opening HTML tag with optional attributes

Declaration: `procedure TagStart(const Name: string; const Attrs: string)`

Visibility: public

Description: Writes an opening HTML tag to the output stream. If attributes are provided, they are included in the tag. This is a utility method used by other formatting methods to generate HTML markup.

See also: THTMLWriter.TagEnd (1632)

68.4.5 THTMLWriter.TagEnd

Synopsis: Writes a closing HTML tag

Declaration: `procedure TagEnd(const Name: string)`

Visibility: public

Description: Writes a closing HTML tag to the output stream. This is a utility method used by other formatting methods to generate HTML markup.

See also: THTMLWriter.TagStart (1632)

68.4.6 THTMLWriter.EscapeText

Synopsis: Escapes HTML special characters

Declaration: `function EscapeText(AText: string) : string;  Override`

Visibility: public

Description: Escapes HTML special characters in text to prevent them from being interpreted as markup. Replaces ampersand (&), less-than (<), greater-than (>) characters with their HTML entities, and converts line feeds to HTML line breaks (<BR>).

See also: TFormattingWriter.EscapeText (1630)

68.4.7 THTMLWriter.DocumentStart

Synopsis: Begins an HTML document

Declaration: `procedure DocumentStart(const Title: string);  Override`

Visibility: public

Description: Begins an HTML document by writing the HTML, TITLE, and BODY opening tags. The title is written into the HTML TITLE element.

See also: THTMLWriter.DocumentEnd (1633), TFormattingWriter.DocumentStart (1630)

68.4.8 THTMLWriter.DocumentEnd

Synopsis: Ends the HTML document

Declaration: `procedure DocumentEnd;  Override`

Visibility: public

Description: Ends the HTML document by writing the BODY and HTML closing tags.

See also: THTMLWriter.DocumentStart (1633), TFormattingWriter.DocumentEnd (1630)

68.4.9 THTMLWriter.HeaderStart

Synopsis: Begins HTML header tag at specified level

Declaration: `procedure HeaderStart(Alevel: Integer);  Override`

Visibility: public

Description: Begins an HTML header by writing the appropriate H1-H6 tag based on the level. Level 1 generates H1, level 2 generates H2, etc.

See also: THTMLWriter.HeaderEnd (1633), TFormattingWriter.HeaderStart (1630)

68.4.10 THTMLWriter.HeaderEnd

Synopsis: Ends HTML header tag at specified level

Declaration: `procedure HeaderEnd(Alevel: Integer);  Override`

Visibility: public

Description: Ends an HTML header by writing the appropriate closing H1-H6 tag based on the level.

See also: THTMLWriter.HeaderStart (1633), TFormattingWriter.HeaderEnd (1630)

68.4.11 THTMLWriter.ParagraphStart

Synopsis: Begins HTML paragraph tag

Declaration: `procedure ParagraphStart;  Override`

Visibility: public

Description: Begins an HTML paragraph by writing the P opening tag.

See also: THTMLWriter.ParagraphEnd (1634), TFormattingWriter.ParagraphStart (1630)

68.4.12 THTMLWriter.ParagraphEnd

Synopsis: Ends HTML paragraph tag

Declaration: `procedure ParagraphEnd;  Override`

Visibility: public

Description: Ends an HTML paragraph by writing the P closing tag.

See also: THTMLWriter.ParagraphStart (1633), TFormattingWriter.ParagraphEnd (1630)

68.4.13 THTMLWriter.LineBreak

Synopsis: Inserts HTML line break tag

Declaration: `procedure LineBreak;  Override`

Visibility: public

Description: Inserts an HTML line break by writing the BR tag.

See also: TFormattingWriter.LineBreak (1630)

68.4.14 THTMLWriter.Rule

Synopsis: Inserts HTML horizontal rule tag

Declaration: `procedure Rule;  Override`

Visibility: public

Description: Inserts an HTML horizontal rule by writing the HR tag.

See also: TFormattingWriter.Rule (1630)

68.4.15 THTMLWriter.BoldStart

Synopsis: Begins HTML bold formatting

Declaration: `procedure BoldStart;  Override`

Visibility: public

Description: Begins HTML bold text formatting by writing the B opening tag.

See also: THTMLWriter.BoldEnd (1634), TFormattingWriter.BoldStart (1630)

68.4.16 THTMLWriter.BoldEnd

Synopsis: Ends HTML bold formatting

Declaration: `procedure BoldEnd;  Override`

Visibility: public

Description: Ends HTML bold text formatting by writing the B closing tag.

See also: THTMLWriter.BoldStart (1634), TFormattingWriter.BoldEnd (1630)

68.4.17 THTMLWriter.ItalicStart

Synopsis: Begins HTML italic formatting

Declaration: `procedure ItalicStart; Override`

Visibility: public

Description: Begins HTML italic text formatting by writing the I opening tag.

See also: THTMLWriter.ItalicEnd (1635), TFormattingWriter.ItalicStart (1630)

68.4.18 THTMLWriter.ItalicEnd

Synopsis: Ends HTML italic formatting

Declaration: `procedure ItalicEnd; Override`

Visibility: public

Description: Ends HTML italic text formatting by writing the I closing tag.

See also: THTMLWriter.ItalicStart (1635), TFormattingWriter.ItalicEnd (1630)

68.4.19 THTMLWriter.UnderlineStart

Synopsis: Begins HTML underline formatting

Declaration: `procedure UnderlineStart; Override`

Visibility: public

Description: Begins HTML underline text formatting by writing the U opening tag.

See also: THTMLWriter.UnderlineEnd (1635), TFormattingWriter.UnderlineStart (1630)

68.4.20 THTMLWriter.UnderlineEnd

Synopsis: Ends HTML underline formatting

Declaration: `procedure UnderlineEnd; Override`

Visibility: public

Description: Ends HTML underline text formatting by writing the U closing tag.

See also: THTMLWriter.UnderlineStart (1635), TFormattingWriter.UnderlineEnd (1630)

68.4.21 THTMLWriter.PreformatStart

Synopsis: Begins HTML preformatted text block

Declaration: `procedure PreformatStart; Override`

Visibility: public

Description: Begins an HTML preformatted text block by writing the PRE opening tag. Text within this block preserves whitespace and line breaks.

See also: THTMLWriter.PreformatEnd (1636), TFormattingWriter.PreformatStart (1630)

68.4.22 THTMLWriter.PreformatEnd

Synopsis: Ends HTML preformatted text block

Declaration: `procedure PreformatEnd;  Override`

Visibility: public

Description: Ends an HTML preformatted text block by writing the PRE closing tag.

See also: THTMLWriter.PreformatStart (1635), TFormattingWriter.PreformatEnd (1630)

68.4.23 THTMLWriter.TableStart

Synopsis: Begins HTML table with optional border

Declaration: `procedure TableStart(NoCols: Integer; Border: Boolean);  Override`

Visibility: public

Description: Begins an HTML table by writing the TABLE opening tag. If border is `True`, includes the BORDER=1 attribute.

See also: THTMLWriter.TableEnd (1636), TFormattingWriter.TableStart (1630)

68.4.24 THTMLWriter.TableEnd

Synopsis: Ends HTML table

Declaration: `procedure TableEnd;  Override`

Visibility: public

Description: Ends an HTML table by writing the TABLE closing tag.

See also: THTMLWriter.TableStart (1636), TFormattingWriter.TableEnd (1630)

68.4.25 THTMLWriter.RowStart

Synopsis: Begins HTML table row

Declaration: `procedure RowStart;  Override`

Visibility: public

Description: Begins an HTML table row by writing the TR opening tag.

See also: THTMLWriter.RowEnd (1636), TFormattingWriter.RowStart (1630)

68.4.26 THTMLWriter.RowEnd

Synopsis: Ends HTML table row

Declaration: `procedure RowEnd;  Override`

Visibility: public

Description: Ends an HTML table row by writing the TR closing tag.

See also: THTMLWriter.RowStart (1636), TFormattingWriter.RowEnd (1630)

68.4.27 THTMLWriter.CellStart

Synopsis: Begins HTML table cell

Declaration: `procedure CellStart;  Override`

Visibility: public

Description: Begins an HTML table cell by writing the TD opening tag.

See also: THTMLWriter.CellEnd (1637), TFormattingWriter.CellStart (1630)

68.4.28 THTMLWriter.CellEnd

Synopsis: Ends HTML table cell

Declaration: `procedure CellEnd;  Override`

Visibility: public

Description: Ends an HTML table cell by writing the TD closing tag.

See also: THTMLWriter.CellStart (1637), TFormattingWriter.CellEnd (1630)

68.4.29 THTMLWriter.HeaderCellStart

Synopsis: Begins HTML table header cell

Declaration: `procedure HeaderCellStart;  Override`

Visibility: public

Description: Begins an HTML table header cell by writing the TH opening tag.

See also: THTMLWriter.HeaderCellEnd (1637), TFormattingWriter.HeaderCellStart (1630)

68.4.30 THTMLWriter.HeaderCellEnd

Synopsis: Ends HTML table header cell

Declaration: `procedure HeaderCellEnd;  Override`

Visibility: public

Description: Ends an HTML table header cell by writing the TH closing tag.

See also: THTMLWriter.HeaderCellStart (1637), TFormattingWriter.HeaderCellEnd (1630)

68.4.31 THTMLWriter.ListStart

Synopsis: Begins HTML list of specified type

Declaration: `procedure ListStart(ListType: TlistType);  Override`

Visibility: public

Description: Begins an HTML list by writing the appropriate list tag. Uses OL for numbered lists, UL for ordered lists, and DL for definition lists based on the ListTags (1630) array.

See also: THTMLWriter.ListEnd (1638), TFormattingWriter.ListStart (1630), ListTags (1630)

68.4.32 THTMLWriter.ListEnd

Synopsis: Ends HTML list of specified type

Declaration:
```
procedure ListEnd(ListType: TlistType);  Override
```

Visibility: public

Description: Ends an HTML list by writing the appropriate closing list tag. Uses the ListTags (1630) array to determine the correct closing tag.

See also: THTMLWriter.ListStart (1637), TFormattingWriter.ListEnd (1630), ListTags (1630)

68.4.33 THTMLWriter.ListItemStart

Synopsis: Begins HTML list item

Declaration:
```
procedure ListItemStart;  Override
```

Visibility: public

Description: Begins an HTML list item by writing the LI opening tag.

See also: THTMLWriter.ListItemEnd (1638), TFormattingWriter.ListItemStart (1630)

68.4.34 THTMLWriter.ListItemEnd

Synopsis: Ends HTML list item

Declaration:
```
procedure ListItemEnd;  Override
```

Visibility: public

Description: Ends an HTML list item by writing the LI closing tag.

See also: THTMLWriter.ListItemStart (1638), TFormattingWriter.ListItemEnd (1630)

68.4.35 THTMLWriter.DefinitionItem

Synopsis: Creates HTML definition list item

Declaration:
```
procedure DefinitionItem(const Aname: string; const AText: string)
                                ;  Override
```

Visibility: public

Description: Creates an HTML definition list item by writing DT tags around the name and DD tags around the description text.

See also: TFormattingWriter.DefinitionItem (1630)

68.4.36 THTMLWriter.FormStart

Synopsis: Begins HTML form with action and method

Declaration:
```
procedure FormStart(const Action: string; const Method: string)
```

Visibility: public

Description: Begins an HTML form by writing the FORM opening tag with ACTION and optional METHOD attributes.

See also: THTMLWriter.FormEnd (1639)

68.4.37 THTMLWriter.FormEnd

Synopsis: Ends HTML form

Declaration:
```
procedure FormEnd
```

Visibility: public

Description: Ends an HTML form by writing the FORM closing tag.

See also: THTMLWriter.FormStart (1639)

68.4.38 THTMLWriter.EmitInput

Synopsis: Emits HTML input field

Declaration:
```
procedure EmitInput(const Name: string; const Value: string)
procedure EmitInput(const Name: string; const Value: string;
                    const Attrs: string)
```

Visibility: public

Description: Emits an HTML INPUT tag with NAME and VALUE attributes. This is an overloaded method that can include additional attributes.

See also: THTMLWriter.EmitPasswordInput (1639), THTMLWriter.EmitCheckBox (1640), THTMLWriter.EmitRadioButton (1640)

68.4.39 THTMLWriter.EmitPasswordInput

Synopsis: Emits HTML password input field

Declaration:
```
procedure EmitPasswordInput(const Name: string; const Value: string)
```

Visibility: public

Description: Emits an HTML INPUT tag with TYPE="password" attribute for secure password entry.

See also: THTMLWriter.EmitInput (1639)

68.4.40 THTMLWriter.EmitCheckBox

Synopsis: Emits HTML checkbox input

Declaration:
```
procedure EmitCheckBox(const Name: string; const Value: string)
procedure EmitCheckBox(const Name: string; const Value: string;
                       Checked: Boolean)
```

Visibility: public

Description: Emits an HTML INPUT tag with TYPE="checkbox" attribute. This is an overloaded method that can specify whether the checkbox is initially checked.

See also: THTMLWriter.EmitRadioButton (1640), THTMLWriter.EmitInput (1639)

68.4.41 THTMLWriter.EmitRadioButton

Synopsis: Emits HTML radio button input

Declaration:
```
procedure EmitRadioButton(const Name: string; const Value: string)
procedure EmitRadioButton(const Name: string; const Value: string;
                          Checked: Boolean)
```

Visibility: public

Description: Emits an HTML INPUT tag with TYPE="radio" attribute. This is an overloaded method that can specify whether the radio button is initially selected.

See also: THTMLWriter.EmitCheckBox (1640), THTMLWriter.EmitInput (1639)

68.4.42 THTMLWriter.EmitArea

Synopsis: Emits HTML textarea for multi-line input

Declaration:
```
procedure EmitArea(const Name: string; const Value: string;
                   Rows: Integer; Cols: Integer)
```

Visibility: public

Description: Emits an HTML TEXTAREA tag with specified rows and columns dimensions. The value is written as the content between opening and closing TEXTAREA tags.

See also: THTMLWriter.EmitInput (1639)

68.4.43 THTMLWriter.EmitComboBox

Synopsis: Emits HTML select dropdown list

Declaration:
```
procedure EmitComboBox(const Name: string; const Value: string;
                       Items: TStrings; UseValues: Boolean)
procedure EmitComboBox(const Name: string; const Value: string;
                       Items: TStrings)
```

Visibility: public

Description: Emits an HTML SELECT tag with OPTION elements from the provided string list. This is an overloaded method that can use name=value pairs when UseValues is `True`.

68.4.44 THTMLWriter.EmitButton

Synopsis: Emits HTML button input

Declaration:
```
procedure EmitButton(const Name: string; const ButtonType: string;
                     const Value: string)
```

Visibility: public

Description: Emits an HTML INPUT tag with the specified button type (submit, reset, button, etc.).

See also: THTMLWriter.EmitSubmitButton (1641), THTMLWriter.EmitResetButton (1641)

68.4.45 THTMLWriter.EmitSubmitButton

Synopsis: Emits HTML submit button

Declaration:
```
procedure EmitSubmitButton(const Name: string; const Value: string)
```

Visibility: public

Description: Emits an HTML INPUT tag with TYPE="submit" for form submission.

See also: THTMLWriter.EmitButton (1641), THTMLWriter.EmitResetButton (1641)

68.4.46 THTMLWriter.EmitResetButton

Synopsis: Emits HTML reset button

Declaration:
```
procedure EmitResetButton(const Name: string; const Value: string)
```

Visibility: public

Description: Emits an HTML INPUT tag with TYPE="reset" for clearing form fields.

See also: THTMLWriter.EmitButton (1641), THTMLWriter.EmitSubmitButton (1641)

68.4.47 THTMLWriter.EmitHiddenVar

Synopsis: Emits HTML hidden input field

Declaration:
```
procedure EmitHiddenVar(const Name: string; const Value: string)
```

Visibility: public

Description: Emits an HTML INPUT tag with TYPE="hidden" for storing invisible form data.

See also: THTMLWriter.EmitInput (1639)

Chapter 69

Reference for unit 'wtex'

69.1 Used units

Table 69.1: Used units by unit 'wtex'

Name	Page
Classes	??
System	??
sysutils	??
wformat	1617

69.2 Overview

The `wtex` unit provides TTexWriter (1643), a concrete implementation of TFormattingWriter (1642) that generates LaTeX output. This class implements all the abstract formatting methods to produce valid LaTeX markup for documents, tables, and lists using standard LaTeX commands and environments.

69.3 Constants, types and variables

69.3.1 Constants

```
ListNames : Array[TListType] of string = ('enumerate', 'itemize',
  'definition')
```

`ListNames` is an array constant that maps TListType (1642) enumeration values to their corresponding LaTeX list environment names. Contains 'enumerate' for numbered lists, 'itemize' for ordered lists, and 'definition' for definition lists.

69.4 TTexWriter

69.4.1 Description

`TTexWriter` is a concrete implementation of TFormattingWriter (1642) that generates LaTeX output. It implements all formatting methods to produce valid LaTeX markup including document structure, text formatting, tables, and lists. The class properly escapes LaTeX special characters and uses standard LaTeX commands and environments.

See also: TFormattingWriter (1642), ListNames (1642)

69.4.2 Method overview

Page	Method	Description
1646	BoldEnd	Ends LaTeX bold formatting
1646	BoldStart	Begins LaTeX bold formatting
1648	CellStart	Begins LaTeX table cell
1650	DefinitionItem	Creates LaTeX definition list item
1645	DocumentEnd	Ends LaTeX document
1645	DocumentStart	Begins LaTeX document
1644	EnvironmentEnd	Ends LaTeX environment
1644	EnvironmentStart	Begins LaTeX environment with optional parameters
1644	EscapeText	Escapes LaTeX special characters
1649	HeaderCellEnd	Ends LaTeX table header cell
1649	HeaderCellStart	Begins LaTeX table header cell
1645	HeaderEnd	Ends LaTeX section header
1645	HeaderStart	Begins LaTeX section header
1647	ItalicEnd	Ends LaTeX italic formatting
1646	ItalicStart	Begins LaTeX italic formatting
1646	LineBreak	Inserts LaTeX line break
1649	ListEnd	Ends LaTeX list environment
1650	ListItemStart	Begins LaTeX list item
1649	ListStart	Begins LaTeX list environment
1645	ParagraphEnd	Ends LaTeX paragraph
1647	PreformatEnd	Ends LaTeX verbatim environment
1647	PreformatStart	Begins LaTeX verbatim environment
1648	RowEnd	Ends LaTeX table row
1648	RowStart	Begins LaTeX table row
1646	Rule	Inserts LaTeX horizontal rule
1644	ScopeEnd	Writes LaTeX closing brace
1643	ScopeStart	Writes LaTeX opening brace
1648	TableEnd	Ends LaTeX tabular environment
1648	TableStart	Begins LaTeX tabular environment
1647	UnderlineEnd	Ends LaTeX underline formatting
1647	UnderlineStart	Begins LaTeX underline formatting

69.4.3 TTexWriter.ScopeStart

Synopsis: Writes LaTeX opening brace

Declaration: `procedure ScopeStart`

Visibility: public

Description: Writes an opening brace ({) to begin a LaTeX scope or command argument. This is a utility method used by other formatting methods to create properly scoped LaTeX constructs.

See also: TTexWriter.ScopeEnd (1644)

69.4.4 TTexWriter.ScopeEnd

Synopsis: Writes LaTeX closing brace

Declaration: `procedure ScopeEnd`

Visibility: public

Description: Writes a closing brace (}) to end a LaTeX scope or command argument. This is a utility method used by other formatting methods to create properly scoped LaTeX constructs.

See also: TTexWriter.ScopeStart (1643)

69.4.5 TTexWriter.EnvironmentStart

Synopsis: Begins LaTeX environment with optional parameters

Declaration: `procedure EnvironmentStart(const Name: string; const Opts: string)`

Visibility: public

Description: Writes a LaTeX \begin command for the specified environment name. If options are provided, they are included before the environment name. This is a utility method for creating LaTeX environments.

See also: TTexWriter.EnvironmentEnd (1644)

69.4.6 TTexWriter.EnvironmentEnd

Synopsis: Ends LaTeX environment

Declaration: `procedure EnvironmentEnd(const Name: string)`

Visibility: public

Description: Writes a LaTeX \end command for the specified environment name. This is a utility method for properly closing LaTeX environments.

See also: TTexWriter.EnvironmentStart (1644)

69.4.7 TTexWriter.EscapeText

Synopsis: Escapes LaTeX special characters

Declaration: `function EscapeText(AText: string) : string;  Override`

Visibility: public

Description: Escapes LaTeX special characters in text to prevent them from being interpreted as commands. Currently replaces underscore (_) characters with escaped underscores (_).

See also: TFormattingWriter.EscapeText (1642)

69.4.8 TTexWriter.DocumentStart

Synopsis: Begins LaTeX document

Declaration: `procedure DocumentStart(const Title: string);  Override`

Visibility: public

Description: Begins a LaTeX document by writing the document class, package includes, document environment, and title command. Generates \documentclass{report}, \usepackage{a4}, \begin{document}, and \title{} markup.

See also: TTexWriter.DocumentEnd (1645), TFormattingWriter.DocumentStart (1642)

69.4.9 TTexWriter.DocumentEnd

Synopsis: Ends LaTeX document

Declaration: `procedure DocumentEnd;  Override`

Visibility: public

Description: Ends a LaTeX document by writing the \end{document} command.

See also: TTexWriter.DocumentStart (1645), TFormattingWriter.DocumentEnd (1642)

69.4.10 TTexWriter.HeaderStart

Synopsis: Begins LaTeX section header

Declaration: `procedure HeaderStart(Alevel: Integer);  Override`

Visibility: public

Description: Begins a LaTeX section header by writing the appropriate sectioning command based on the level. Level 0 generates \part, level 1 generates \chapter, level 2 generates \section, etc.

See also: TTexWriter.HeaderEnd (1645), TFormattingWriter.HeaderStart (1642)

69.4.11 TTexWriter.HeaderEnd

Synopsis: Ends LaTeX section header

Declaration: `procedure HeaderEnd(Alevel: Integer);  Override`

Visibility: public

Description: Ends a LaTeX section header by writing the closing brace and adding a blank line.

See also: TTexWriter.HeaderStart (1645), TFormattingWriter.HeaderEnd (1642)

69.4.12 TTexWriter.ParagraphEnd

Synopsis: Ends LaTeX paragraph

Declaration: `procedure ParagraphEnd;  Override`

Visibility: public

Description: Ends a LaTeX paragraph by writing two line feeds to create a paragraph break.

See also: TFormattingWriter.ParagraphEnd (1642)

69.4.13 TTexWriter.LineBreak

Synopsis: Inserts LaTeX line break

Declaration: `procedure LineBreak; Override`

Visibility: public

Description: Inserts a LaTeX line break by writing the \\ command.

See also: TFormattingWriter.LineBreak (1642)

69.4.14 TTexWriter.Rule

Synopsis: Inserts LaTeX horizontal rule

Declaration: `procedure Rule; Override`

Visibility: public

Description: Inserts a LaTeX horizontal rule by writing the \hline command.

See also: TFormattingWriter.Rule (1642)

69.4.15 TTexWriter.BoldStart

Synopsis: Begins LaTeX bold formatting

Declaration: `procedure BoldStart; Override`

Visibility: public

Description: Begins LaTeX bold text formatting by writing the \textbf command and opening brace.

See also: TTexWriter.BoldEnd (1646), TFormattingWriter.BoldStart (1642)

69.4.16 TTexWriter.BoldEnd

Synopsis: Ends LaTeX bold formatting

Declaration: `procedure BoldEnd; Override`

Visibility: public

Description: Ends LaTeX bold text formatting by writing the closing brace.

See also: TTexWriter.BoldStart (1646), TFormattingWriter.BoldEnd (1642)

69.4.17 TTexWriter.ItalicStart

Synopsis: Begins LaTeX italic formatting

Declaration: `procedure ItalicStart; Override`

Visibility: public

Description: Begins LaTeX italic text formatting by writing the \textit command and opening brace.

See also: TTexWriter.ItalicEnd (1647), TFormattingWriter.ItalicStart (1642)

69.4.18 TTexWriter.ItalicEnd

Synopsis: Ends LaTeX italic formatting

Declaration: `procedure ItalicEnd; Override`

Visibility: public

Description: Ends LaTeX italic text formatting by writing the closing brace.

See also: TTexWriter.ItalicStart (1646), TFormattingWriter.ItalicEnd (1642)

69.4.19 TTexWriter.UnderlineStart

Synopsis: Begins LaTeX underline formatting

Declaration: `procedure UnderlineStart; Override`

Visibility: public

Description: Begins LaTeX underline text formatting by writing the \textul command and opening brace.

See also: TTexWriter.UnderlineEnd (1647), TFormattingWriter.UnderlineStart (1642)

69.4.20 TTexWriter.UnderlineEnd

Synopsis: Ends LaTeX underline formatting

Declaration: `procedure UnderlineEnd; Override`

Visibility: public

Description: Ends LaTeX underline text formatting by writing the closing brace.

See also: TTexWriter.UnderlineStart (1647), TFormattingWriter.UnderlineEnd (1642)

69.4.21 TTexWriter.PreformatStart

Synopsis: Begins LaTeX verbatim environment

Declaration: `procedure PreformatStart; Override`

Visibility: public

Description: Begins a LaTeX verbatim environment where text is displayed exactly as entered with fixed-width font and preserved formatting.

See also: TTexWriter.PreformatEnd (1647), TFormattingWriter.PreformatStart (1642)

69.4.22 TTexWriter.PreformatEnd

Synopsis: Ends LaTeX verbatim environment

Declaration: `procedure PreformatEnd; Override`

Visibility: public

Description: Ends a LaTeX verbatim environment.

See also: TTexWriter.PreformatStart (1647), TFormattingWriter.PreformatEnd (1642)

69.4.23 TTexWriter.TableStart

Synopsis: Begins LaTeX tabular environment

Declaration: `procedure TableStart(NoCols: Integer; Border: Boolean);  Override`

Visibility: public

Description: Begins a LaTeX tabular environment with column specifications. Generates column specification with left-aligned columns (l) based on the number of columns specified.

See also: TTexWriter.TableEnd (1648), TFormattingWriter.TableStart (1642)

69.4.24 TTexWriter.TableEnd

Synopsis: Ends LaTeX tabular environment

Declaration: `procedure TableEnd;  Override`

Visibility: public

Description: Ends a LaTeX tabular environment.

See also: TTexWriter.TableStart (1648), TFormattingWriter.TableEnd (1642)

69.4.25 TTexWriter.RowStart

Synopsis: Begins LaTeX table row

Declaration: `procedure RowStart;  Override`

Visibility: public

Description: Begins a LaTeX table row by resetting the cell counter to zero.

See also: TTexWriter.RowEnd (1648), TFormattingWriter.RowStart (1642)

69.4.26 TTexWriter.RowEnd

Synopsis: Ends LaTeX table row

Declaration: `procedure RowEnd;  Override`

Visibility: public

Description: Ends a LaTeX table row by writing the row terminator (\\) followed by a line break.

See also: TTexWriter.RowStart (1648), TFormattingWriter.RowEnd (1642)

69.4.27 TTexWriter.CellStart

Synopsis: Begins LaTeX table cell

Declaration: `procedure CellStart;  Override`

Visibility: public

Description: Begins a LaTeX table cell by writing the column separator (&) if this is not the first cell in the row. Increments the internal cell counter.

See also: TTexWriter.FCellCount (**??**), TFormattingWriter.CellStart (1642)

69.4.28 TTexWriter.HeaderCellStart

Synopsis: Begins LaTeX table header cell

Declaration: `procedure HeaderCellStart; Override`

Visibility: public

Description: Begins a LaTeX table header cell by calling CellStart (1648). LaTeX does not distinguish between header and regular cells in the tabular environment.

See also: TTexWriter.CellStart (1648), TTexWriter.HeaderCellEnd (1649), TFormattingWriter.HeaderCellStart (1642)

69.4.29 TTexWriter.HeaderCellEnd

Synopsis: Ends LaTeX table header cell

Declaration: `procedure HeaderCellEnd; Override`

Visibility: public

Description: Ends a LaTeX table header cell by calling CellEnd (1643). LaTeX does not require special handling for header cell endings in the tabular environment.

See also: TTexWriter.CellEnd (1643), TTexWriter.HeaderCellStart (1649), TFormattingWriter.HeaderCellEnd (1642)

69.4.30 TTexWriter.ListStart

Synopsis: Begins LaTeX list environment

Declaration: `procedure ListStart(ListType: TlistType); Override`

Visibility: public

Description: Begins a LaTeX list environment based on the list type. Uses enumerate for numbered lists, itemize for ordered lists, and definition for definition lists based on the ListNames (1642) array.

See also: TTexWriter.ListEnd (1649), TFormattingWriter.ListStart (1642), ListNames (1642)

69.4.31 TTexWriter.ListEnd

Synopsis: Ends LaTeX list environment

Declaration: `procedure ListEnd(ListType: TlistType); Override`

Visibility: public

Description: Ends a LaTeX list environment by writing the appropriate \end command. Uses the ListNames (1642) array to determine the correct environment name.

See also: TTexWriter.ListStart (1649), TFormattingWriter.ListEnd (1642), ListNames (1642)

69.4.32 TTexWriter.ListItemStart

Synopsis: Begins LaTeX list item

Declaration:
```
procedure ListItemStart;  Override
```

Visibility: public

Description: Begins a LaTeX list item by writing the \item command.

See also: TFormattingWriter.ListItemStart (1642)

69.4.33 TTexWriter.DefinitionItem

Synopsis: Creates LaTeX definition list item

Declaration:
```
procedure DefinitionItem(const Aname: string; const AText: string)
                         ;  Override
```

Visibility: public

Description: Creates a LaTeX definition list item by writing \item with the name in square brackets followed by the definition text.

See also: TFormattingWriter.DefinitionItem (1642)

Chapter 70

Reference for unit 'Zipper'

70.1 Used units

Table 70.1: Used units by unit 'Zipper'

Name	Page
BaseUnix	??
Classes	??
System	??
sysutils	??
ZStream	1690

70.2 Overview

zipper implements zip compression/decompression compatible with the popular .ZIP format. The zip file format is documented at:

http://www.pkware.com/documents/casestudies/APPNOTE.TXT.

The Pascal conversion of the standard zlib library was implemented by Jacques Nomssi Nzali. It is used in the FCL to implement the `TCompressionStream` class.

70.3 Constants, types and variables

70.3.1 Constants

```
CENTRAL_FILE_HEADER_SIGNATURE = $02014B50
```

Denotes beginning of a file entry inside the zip directory. A file header follows this marker.

```
Crc_32_Tab : Array[0..255] of LongWord = ($00000000, $77073096, $ee0e612c
  , $990951ba, $076dc419, $706af48f, $e963a535, $9e6495a3, $0edb8832
  , $79dcb8a4, $e0d5e91e, $97d2d988, $09b64c2b, $7eb17cbd, $e7b82d07
  , $90bf1d91, $1db71064, $6ab020f2, $f3b97148, $84be41de, $1adad47d
  , $6ddde4eb, $f4d4b551, $83d385c7, $136c9856, $646ba8c0, $fd62f97a
```

```
  , $8a65c9ec, $14015c4f, $63066cd9, $fa0f3d63, $8d080df5, $3b6e20c8
  , $4c69105e, $d56041e4, $a2677172, $3c03e4d1, $4b04d447, $d20d85fd
  , $a50ab56b, $35b5a8fa, $42b2986c, $dbbbc9d6, $acbcf940, $32d86ce3
  , $45df5c75, $dcd60dcf, $abd13d59, $26d930ac, $51de003a, $c8d75180
  , $bfd06116, $21b4f4b5, $56b3c423, $cfba9599, $b8bda50f, $2802b89e
  , $5f058808, $c60cd9b2, $b10be924, $2f6f7c87, $58684c11, $c1611dab
  , $b6662d3d, $76dc4190, $01db7106, $98d220bc, $efd5102a, $71b18589
  , $06b6b51f, $9fbfe4a5, $e8b8d433, $7807c9a2, $0f00f934, $9609a88e
  , $e10e9818, $7f6a0dbb, $086d3d2d, $91646c97, $e6635c01, $6b6b51f4
  , $1c6c6162, $856530d8, $f262004e, $6c0695ed, $1b01a57b, $8208f4c1
  , $f50fc457, $65b0d9c6, $12b7e950, $8bbeb8ea, $fcb9887c, $62dd1ddf
  , $15da2d49, $8cd37cf3, $fbd44c65, $4db26158, $3ab551ce, $a3bc0074
  , $d4bb30e2, $4adfa541, $3dd895d7, $a4d1c46d, $d3d6f4fb, $4369e96a
  , $346ed9fc, $ad678846, $da60b8d0, $44042d73, $33031de5, $aa0a4c5f
  , $dd0d7cc9, $5005713c, $270241aa, $be0b1010, $c90c2086, $5768b525
  , $206f85b3, $b966d409, $ce61e49f, $5edef90e, $29d9c998, $b0d09822
  , $c7d7a8b4, $59b33d17, $2eb40d81, $b7bd5c3b, $c0ba6cad, $edb88320
  , $9abfb3b6, $03b6e20c, $74b1d29a, $ead54739, $9dd277af, $04db2615
  , $73dc1683, $e3630b12, $94643b84, $0d6d6a3e, $7a6a5aa8, $e40ecf0b
  , $9309ff9d, $0a00ae27, $7d079eb1, $f00f9344, $8708a3d2, $1e01f268
  , $6906c2fe, $f762575d, $806567cb, $196c3671, $6e6b06e7, $fed41b76
  , $89d32be0, $10da7a5a, $67dd4acc, $f9b9df6f, $8ebeeff9, $17b7be43
  , $60b08ed5, $d6d6a3e8, $a1d1937e, $38d8c2c4, $4fdff252, $d1bb67f1
  , $a6bc5767, $3fb506dd, $48b2364b, $d80d2bda, $af0a1b4c, $36034af6
  , $41047a60, $df60efc3, $a867df55, $316e8eef, $4669be79, $cb61b38c
  , $bc66831a, $256fd2a0, $5268e236, $cc0c7795, $bb0b4703, $220216b9
  , $5505262f, $c5ba3bbe, $b2bd0b28, $2bb45a92, $5cb36a04, $c2d7ffa7
  , $b5d0cf31, $2cd99e8b, $5bdeae1d, $9b64c2b0, $ec63f226, $756aa39c
  , $026d930a, $9c0906a9, $eb0e363f, $72076785, $05005713, $95bf4a82
  , $e2b87a14, $7bb12bae, $0cb61b38, $92d28e9b, $e5d5be0d, $7cdcefb7
  , $0bdbdf21, $86d3d2d4, $f1d4e242, $68ddb3f8, $1fda836e, $81be16cd
  , $f6b9265b, $6fb077e1, $18b74777, $88085ae6, $ff0f6a70, $66063bca
  , $11010b5c, $8f659eff, $f862ae69, $616bffd3, $166ccf45, $a00ae278
  , $d70dd2ee, $4e048354, $3903b3c2, $a7672661, $d06016f7, $4969474d
  , $3e6e77db, $aed16a4a, $d9d65adc, $40df0b66, $37d83bf0, $a9bcae53
  , $debb9ec5, $47b2cf7f, $30b5ffe9, $bdbdf21c, $cabac28a, $53b39330
  , $24b4a3a6, $bad03605, $cdd70693, $54de5729, $23d967bf, $b3667a2e
  , $c4614ab8, $5d681b02, $2a6f2b94, $b40bbe37, $c30c8ea1, $5a05df1b
  , $2d02ef8d)
```

Table used in determining CRC-32 values. There are various CRC-32 algorithms in use; please refer to the ZIP file format specifications for details.

```
EFS_LANGUAGE_ENCODING_FLAG = $800
```

Language encoding flag (EFS). When set the file name and comment fields must use UTF-8 encoding.

```
END_OF_CENTRAL_DIR_SIGNATURE = $06054B50
```

Marker specifying end of directory within zip file.

```
FIRSTENTRY = 257
```

Offset of First entry in table.

```
INFOZIP_UNICODE_PATH_ID = $7075
```

```
LOCAL_FILE_HEADER_SIGNATURE = $04034B50
```

Denotes beginning of a file header within the zip file. A file header follows this marker, followed by the file data proper.

```
OS_FAT = 0
```

MS-DOS and OS/2 (FAT/VFAT/FAT32).

```
OS_NTFS = 10
```

NTFS.

```
OS_OS2 = 6
```

OS/2 HPFS.

```
OS_OSX = 19
```

Mac OSX.

```
OS_UNIX = 3
```

UNIX-like platforms.

```
OS_VFAT = 14
```

VFAT.

```
TABLESIZE = 8191
```

Size for the code table used in LZW compression.

```
UNIX_BLK = $6000
```

Unix block device.

```
UNIX_CHAR = $2000
```

Unix character device.

```
UNIX_DEFAULT = UNIX_RUSR or UNIX_WUSR or UNIX_XUSR or UNIX_RGRP or
  UNIX_ROTH
```

Unix default attributes.

```
UNIX_DIR = $4000
```

Unix directory.

```
UNIX_FIFO = $1000
```

Unix FIFO file type.

```
UNIX_FILE = $8000
```

Unix regular file.

```
UNIX_LINK = $A000
```

Unix symbolic link.

```
UNIX_MASK = $F000
```

Unix permission mask.

```
UNIX_RGRP = $0020
```

Unix group read permission.

```
UNIX_ROTH = $0004
```

Unix other users read permission.

```
UNIX_RUSR = $0100
```

Unix user read permission.

```
UNIX_SOCK = $C000
```

Unix sockets.

```
UNIX_WGRP = $0010
```

Unix group write permission.

```
UNIX_WOTH = $0002
```

Unix other users write permission.

```
UNIX_WUSR = $0080
```

Unix user write permission.

```
UNIX_XGRP = $0008
```

Unix group execute permission.

```
UNIX_XOTH = $0001
```

Unix other users execute permission.

```
UNIX_XUSR = $0040
```

Unix user execute permission.

```
ZIP64_END_OF_CENTRAL_DIR_LOCATOR_SIGNATURE = $07064B50
```

```
ZIP64_END_OF_CENTRAL_DIR_SIGNATURE = $06064B50
```

Marker specifying end of the directory within a 64-bit zip file.

```
ZIP64_HEADER_ID = $0001
```

70.3.2 Types

```
BufPtr = PByte
```

Alias for the PByte type. Used to implement the output buffer in TShrinker.

```
CodeArray = Array[0..TABLESIZE] of CodeRec
```

Array definition for CodeRec (1657)

```
FreeListArray = Array[FIRSTENTRY..TABLESIZE] of Word
```

Helper type in decoding the zip file.

```
FreeListPtr = ^FreeListArray
```

Pointer to FreeListArray (1655)

```
TablePtr = ^CodeArray
```

Pointer to CodeArray (1655)

```
TCustomInputStreamEvent = procedure(Sender: TObject;
  var AStream: TStream) of object
```

Specifies an event handler signalled for actions to an input stream.

```
TOnCustomStreamEvent = procedure(Sender: TObject; var AStream: TStream
  ;
                                 AItem: TFullZipFileEntry) of
  object
```

Specifies an event handler signalled for stream actions in TUnZipper.

```
TOnEndOfFileEvent = procedure(Sender: TObject; const Ratio: Double
  )
                                of object
```

Event procedure for an end of file (de)compression event.

```
TOnStartFileEvent = procedure(Sender: TObject;
  const AFileName: AnsiString) of object
```

Event procedure for a start of file (de)compression event.

```
TProgressEvent = procedure(Sender: TObject; const Pct: Double) of
  object
```

Event procedure for capturing compression/decompression progress.

```
TProgressEventEx = procedure(Sender: TObject; const ATotPos: Int64
  ;
                              const ATotSize: Int64) of object
```

TProgressEventEx is an object procedure which implements an event handler signalled to indicate compression/decompression progress. It is very similar to TProgressEvent, but provides separate values for the cumulative number of bytes handled and the total number of bytes to be processed.

TProgressEventEx is the type used to implement the OnProgressEx property in TDeCompressor and TUnZipper.

70.4 Central_File_Header_Type

```
Central_File_Header_Type = packed record
  Signature : LongInt;
  MadeBy_Version : Word;
  Extract_Version_Reqd : Word;
  Bit_Flag
  : Word;
  Compress_Method : Word;
  Last_Mod_Time : Word;
  Last_Mod_Date
  : Word;
  Crc32 : LongWord;
  Compressed_Size : LongWord;
  Uncompressed_Size
  : LongWord;
  Filename_Length : Word;
  Extra_Field_Length : Word
  ;
  File_Comment_Length : Word;
  Starting_Disk_Num : Word;
  Internal_Attributes
  : Word;
  External_Attributes : LongWord;
```

```
  Local_Header_Offset
  : LongWord;
end
```

This record contains the structure for a file header within the central directory.

70.5 CodeRec

```
CodeRec = packed record
  Child : SmallInt;
  Sibling : SmallInt;
  Suffix : Byte;
end
```

Small LZW compression helper type.

70.6 End_of_Central_Dir_Type

```
End_of_Central_Dir_Type = packed record
  Signature : LongInt;
  Disk_Number
  : Word;
  Central_Dir_Start_Disk : Word;
  Entries_This_Disk : Word
  ;
  Total_Entries : Word;
  Central_Dir_Size : LongWord;
  Start_Disk_Offset
  : LongWord;
  ZipFile_Comment_Length : Word;
end
```

The end of central directory is placed at the end of the zip file. Note that the end of central directory record is distinct from the Zip64 end of central directory record and zip64 end of central directory locator, which precede the end of central directory, if implemented.

70.7 Extensible_Data_Field_Header_Type

```
Extensible_Data_Field_Header_Type = packed record
  Header_ID : Word
  ;
  Data_Size : Word;
end
```

Beginning of extra field. Occurs after the local file header and after the central directory header.

70.8 Local_File_Header_Type

```
Local_File_Header_Type = packed record
```

```
  Signature : LongInt;
  Extract_Version_Reqd
  : Word;
  Bit_Flag : Word;
  Compress_Method : Word;
  Last_Mod_Time
  : Word;
  Last_Mod_Date : Word;
  Crc32 : LongWord;
  Compressed_Size
  : LongWord;
  Uncompressed_Size : LongWord;
  Filename_Length : Word
  ;
  Extra_Field_Length : Word;
end
```

Record structure containing local file header.

70.9 Zip64_End_of_Central_Dir_Locator_type

```
Zip64_End_of_Central_Dir_Locator_type = packed record
  Signature
  : LongInt;
  Zip64_EOCD_Start_Disk : LongWord;
  Central_Dir_Zip64_EOCD_Offset
  : QWord;
  Total_Disks : LongWord;
end
```

Comes after the Zip64_End_of_Central_Dir_type.

70.10 Zip64_End_of_Central_Dir_type

```
Zip64_End_of_Central_Dir_type = packed record
  Signature : LongInt
  ;
  Record_Size : QWord;
  Version_Made_By : Word;
  Extract_Version_Reqd
  : Word;
  Disk_Number : LongWord;
  Central_Dir_Start_Disk : LongWord
  ;
  Entries_This_Disk : QWord;
  Total_Entries : QWord;
  Central_Dir_Size
  : QWord;
  Start_Disk_Offset : QWord;
end
```

This record appears at the end of the central directory

70.11 Zip64_Extended_Info_Field_Type

```
Zip64_Extended_Info_Field_Type = packed record
  Original_Size : QWord
  ;
  Compressed_Size : QWord;
  Relative_Hdr_Offset : QWord;
  Disk_Start_Number
  : LongWord;
end
```

70.12 EZipError

70.12.1 Description

Exception raised for errors in TZipper and TUnZipper.

70.13 TCompressor

70.13.1 Description

This object compresses a stream into a compressed zip stream.

70.13.2 Method overview

Page	Method	Description
1660	Compress	Compresses input stream to output stream.
1659	Create	Creates a TCompressor (1659) object.
1660	Terminate	Halts the compressor by setting the Terminated property to True.
1660	ZipBitFlag	Current bit.
1660	ZipID	Identifier for type of compression.
1660	ZipVersionReqd	ZIP version required in the method.

70.13.3 Property overview

Page	Properties	Access	Description
1660	BufferSize	r	Size of the buffer used for compression.
1661	Crc32Val	rw	Running CRC32 value.
1661	OnPercent	rw	Threshold percentage which triggers an OnProgress update.
1661	OnProgress	rw	Event handler signalled to indicate the completion percentage for the compressor.
1661	Terminated	r	Set to True when the Terminate method is called.

70.13.4 TCompressor.Create

Synopsis: Creates a TCompressor (1659) object.

Declaration:
```
constructor Create(AInFile: TStream; AOutFile: TStream;
                   ABufSize: LongWord);  Virtual
```

Visibility: public

70.13.5 TCompressor.Compress

Synopsis: Compresses input stream to output stream.

Declaration: `procedure Compress;  Virtual;  Abstract`

Visibility: public

70.13.6 TCompressor.ZipID

Synopsis: Identifier for type of compression.

Declaration: `class function ZipID : Word;  Virtual;  Abstract`

Visibility: public

70.13.7 TCompressor.ZipVersionReqd

Synopsis: ZIP version required in the method.

Declaration: `class function ZipVersionReqd : Word;  Virtual;  Abstract`

Visibility: public

Description: Abstract virtual class function. Must be implemented in a descendent class.

70.13.8 TCompressor.ZipBitFlag

Synopsis: Current bit.

Declaration: `function ZipBitFlag : Word;  Virtual;  Abstract`

Visibility: public

Description: Abstract virtual function. Must be implemented in a descendent class.

70.13.9 TCompressor.Terminate

Synopsis: Halts the compressor by setting the Terminated property to True.

Declaration: `procedure Terminate`

Visibility: public

Description: Halts the compressor by setting the Terminated property to True.

70.13.10 TCompressor.BufferSize

Synopsis: Size of the buffer used for compression.

Declaration: `Property BufferSize : LongWord`

Visibility: public

Access: Read

Description: `BufferSize` is a read-only `LongWord` property with the size of the buffer used for compression. The property is set to the value passed as an argument to the `Create` constructor.

BufferSize is used in the `Compress` method (in descendent classes) to allocate a pointer to a memory block with the required size. It also determines the read size used when processing an input file or stream.

See also: TCompressor.Create (1659), TShrinker.Compress (1669), TDeflater.Compress (1664)

70.13.11 TCompressor.OnPercent

Synopsis: Threshold percentage which triggers an OnProgress update.

Declaration: `Property OnPercent : Integer`

Visibility: public

Access: Read,Write

70.13.12 TCompressor.OnProgress

Synopsis: Event handler signalled to indicate the completion percentage for the compressor.

Declaration: `Property OnProgress : TProgressEvent`

Visibility: public

Access: Read,Write

70.13.13 TCompressor.Crc32Val

Synopsis: Running CRC32 value.

Declaration: `Property Crc32Val : LongWord`

Visibility: public

Access: Read,Write

Description: Running CRC32 value used when writing zip header.

70.13.14 TCompressor.Terminated

Synopsis: Set to True when the Terminate method is called.

Declaration: `Property Terminated : Boolean`

Visibility: public

Access: Read

Description: Set to True when the Terminate method is called.

70.14 TDeCompressor

70.14.1 Description

This object decompresses a compressed zip stream.

70.14.2 Method overview

Page	Method	Description
1662	Create	Creates decompressor object.
1662	DeCompress	Decompress zip stream.
1662	Terminate	Halts decompression and sets Terminated to True.
1662	ZipID	Identifier for type of compression.

70.14.3 Property overview

Page	Properties	Access	Description
1663	BufferSize	r	Size of buffer used in decompression.
1663	Crc32Val	rw	Running CRC32 value used for verifying zip file integrity.
1663	OnPercent	rw	Percentage of decompression completion.
1663	OnProgress	rw	Event handler for OnProgress procedure.
1663	OnProgressEx	rw	Event handler signalled to indicate progress using processed and total byte counts.
1664	Terminated	r	Set to True when the Terminate method is called.

70.14.4 TDeCompressor.Create

Synopsis: Creates decompressor object.

Declaration:
```
constructor Create(AInFile: TStream; AOutFile: TStream;
                   ABufSize: LongWord);  Virtual
```

Visibility: public

70.14.5 TDeCompressor.DeCompress

Synopsis: Decompress zip stream.

Declaration:
```
procedure DeCompress;  Virtual;  Abstract
```

Visibility: public

70.14.6 TDeCompressor.Terminate

Synopsis: Halts decompression and sets Terminated to True.

Declaration:
```
procedure Terminate
```

Visibility: public

Description: Halts decompression and sets Terminated to True.

70.14.7 TDeCompressor.ZipID

Synopsis: Identifier for type of compression.

Declaration:
```
class function ZipID : Word;  Virtual;  Abstract
```

Visibility: public

70.14.8 TDeCompressor.BufferSize

Synopsis: Size of buffer used in decompression.

Declaration: `Property BufferSize : LongWord`

Visibility: public

Access: Read

70.14.9 TDeCompressor.OnPercent

Synopsis: Percentage of decompression completion.

Declaration: `Property OnPercent : Integer`

Visibility: public

Access: Read,Write

70.14.10 TDeCompressor.OnProgress

Synopsis: Event handler for OnProgress procedure.

Declaration: `Property OnProgress : TProgressEvent`

Visibility: public

Access: Read,Write

70.14.11 TDeCompressor.OnProgressEx

Synopsis: Event handler signalled to indicate progress using processed and total byte counts.

Declaration: `Property OnProgressEx : TProgressEventEx`

Visibility: public

Access: Read,Write

Description: Event handler signalled to indicate progress using processed and total byte counts.

70.14.12 TDeCompressor.Crc32Val

Synopsis: Running CRC32 value used for verifying zip file integrity.

Declaration: `Property Crc32Val : LongWord`

Visibility: public

Access: Read,Write

70.14.13 TDeCompressor.Terminated

Synopsis: Set to True when the Terminate method is called.

Declaration:
```
Property Terminated : Boolean
```

Visibility: public

Access: Read

Description: Set to True when the Terminate method is called.

70.15 TDeflater

70.15.1 Description

Child of TCompressor (1659) that implements the Deflate compression method.

70.15.2 Method overview

Page	Method	Description
1664	Compress	Performs compression using the Deflate algorithm.
1664	Create	Constructor for the class instance.
1665	ZipBitFlag	Bitness flag.
1665	ZipID	Zip algorithm ID.
1665	ZipVersionReqd	Required version.

70.15.3 Property overview

Page	Properties	Access	Description
1665	CompressionLevel	rw	Indicates the compression level applied in the Compress method.

70.15.4 TDeflater.Create

Synopsis: Constructor for the class instance.

Declaration:
```
constructor Create(AInFile: TStream; AOutFile: TStream;
                   ABufSize: LongWord);  Override
```

Visibility: public

Description: `Create` is the overridden constructor for the class instance, and calls the inherited method on entry. Create sets the default value for the `CompressionLevel` property to `clNone`.

See also: TDeflater.CompressionLevel (1665), TCompressor.Create (1659)

70.15.5 TDeflater.Compress

Synopsis: Performs compression using the Deflate algorithm.

Declaration:
```
procedure Compress;  Override
```

Visibility: public

Description: Creates a temporary TCompressionStream instance using the compression level specified in the CompressLevel property. Compress signals the OnProgress event handler (when assigned) when the number of bytes representing the OnPercent threshold are processed in the method.

70.15.6 TDeflater.ZipID

Synopsis: Zip algorithm ID.

Declaration: `class function ZipID : Word;  Override`

Visibility: public

Description: Zip algorithm ID.

70.15.7 TDeflater.ZipVersionReqd

Synopsis: Required version.

Declaration: `class function ZipVersionReqd : Word;  Override`

Visibility: public

Description: Required version.

70.15.8 TDeflater.ZipBitFlag

Synopsis: Bitness flag.

Declaration: `function ZipBitFlag : Word;  Override`

Visibility: public

Description: Bitness flag.

70.15.9 TDeflater.CompressionLevel

Synopsis: Indicates the compression level applied in the Compress method.

Declaration: `Property CompressionLevel : Tcompressionlevel`

Visibility: public

Access: Read,Write

Description: `CompressionLevel` is a `TCompressionLevel` property which Indicates the compression level applied in the `Compress` method. Values include:

clNone Do not use compression, just copy data.

clFastest Use the fast (but less) compression.

clDefault Use the default compression. dd

clMax Use the maximum compression.

See also: TDeflater.Compress (1664), TCompressionLevel (1690)

70.16 TFullZipFileEntries

70.16.1 Description

Collection of TFullZipFileEntry items.

See also: TFullZipFileEntry (1666)

70.16.2 Property overview

Page	Properties	Access	Description
1666	FullEntries	rw	Array access to all entries.

70.16.3 TFullZipFileEntries.FullEntries

Synopsis: Array access to all entries.

Declaration: `Property FullEntries[AIndex: Integer]: TFullZipFileEntry; default`

Visibility: public

Access: Read,Write

Description: Array access to all entries.

See also: TFullZipFileEntry (1666)

70.17 TFullZipFileEntry

70.17.1 Description

`TFullZipFileEntry` is a `TZipFileEntry` descendant which provides additional information about files in a .ZIP archive. TFullZipFileEntry extends the ancestor class to include properties like:

BitFlags General purpose bit flag from the Local Header in the .ZIP archive file.

CompressMethod Compression method for the file.

CompressedSize Size after applying the compression method and level.

CRC32 32-bit CRC value for the file.

70.17.2 Property overview

Page	Properties	Access	Description
1667	BitFlags	r	General purpose bit flag from the Local Header in the .ZIP archive file.
1667	CompressedSize	r	Size after applying the compression method and level.
1667	CompressMethod	r	Compression method for the file.
1667	CRC32	rw	32-bit CRC value for the file.

70.17.3 TFullZipFileEntry.BitFlags

Synopsis: General purpose bit flag from the Local Header in the .ZIP archive file.

Declaration: `Property BitFlags : Word`

Visibility: public

Access: Read

Description: General purpose bit flag from the Local Header in the .ZIP archive file.

70.17.4 TFullZipFileEntry.CompressMethod

Synopsis: Compression method for the file.

Declaration: `Property CompressMethod : Word`

Visibility: public

Access: Read

Description: Compression method for the file.

70.17.5 TFullZipFileEntry.CompressedSize

Synopsis: Size after applying the compression method and level.

Declaration: `Property CompressedSize : QWord`

Visibility: public

Access: Read

Description: Size after applying the compression method and level.

70.17.6 TFullZipFileEntry.CRC32

Synopsis: 32-bit CRC value for the file.

Declaration: `Property CRC32 : LongWord`

Visibility: public

Access: Read,Write

Description: 32-bit CRC value for the file.

70.18 TInflater

70.18.1 Description

Child of TDeCompressor (1661) that implements the Inflate decompression method.

70.18.2 Method overview

Page	Method	Description
1668	Create	Constructor for the class instance.
1668	DeCompress	Removes compression applied using the deflate algorithm.
1668	ZipID	Zip algorithm ID.

70.18.3 TInflater.Create

Synopsis: Constructor for the class instance.

Declaration:
```
constructor Create(AInFile: TStream; AOutFile: TStream;
                   ABufSize: LongWord);  Override
```

Visibility: public

Description: Constructor for the class instance.

70.18.4 TInflater.DeCompress

Synopsis: Removes compression applied using the deflate algorithm.

Declaration:
```
procedure DeCompress;  Override
```

Visibility: public

Description: Removes compression applied using the deflate algorithm.

70.18.5 TInflater.ZipID

Synopsis: Zip algorithm ID.

Declaration:
```
class function ZipID : Word;  Override
```

Visibility: public

Description: Zip algorithm ID.

70.19 TShrinker

70.19.1 Description

TShrinker implements the LZW lossless data compression algorithm created by Abraham Lempel, Jacob Ziv, and Terry Welch also known as "shrink" compression.

70.19.2 Method overview

Page	Method	Description
1669	Compress	Compresses input values using LZW (shrink) compression.
1669	Create	Constructor for the class instance.
1669	Destroy	Destructor for the class instance.
1669	ZipBitFlag	Zip bitness flag.
1669	ZipID	Return Zip algorithm ID.
1669	ZipVersionReqd	Minimum zip algorithm required.

70.19.3 TShrinker.Create

Synopsis: Constructor for the class instance.

Declaration:
```
constructor Create(AInFile: TStream; AOutFile: TStream;
                   ABufSize: LongWord);  Override
```

Visibility: public

Description: Constructor for the class instance.

70.19.4 TShrinker.Destroy

Synopsis: Destructor for the class instance.

Declaration:
```
destructor Destroy;  Override
```

Visibility: public

Description: Destructor for the class instance.

70.19.5 TShrinker.Compress

Synopsis: Compresses input values using LZW (shrink) compression.

Declaration:
```
procedure Compress;  Override
```

Visibility: public

Description: Initializes the code table used for LZW compression. Processes buffer-size chunks from the input stream and calls the private Shrink method to generate values written to the output stream.

70.19.6 TShrinker.ZipID

Synopsis: Return Zip algorithm ID.

Declaration:
```
class function ZipID : Word;  Override
```

Visibility: public

Description: Return Zip algorithm ID.

70.19.7 TShrinker.ZipVersionReqd

Synopsis: Minimum zip algorithm required.

Declaration:
```
class function ZipVersionReqd : Word;  Override
```

Visibility: public

Description: Minimum zip algorithm required.

70.19.8 TShrinker.ZipBitFlag

Synopsis: Zip bitness flag.

Declaration:
```
function ZipBitFlag : Word;  Override
```

Visibility: public

Description: Zip bitness flag.

70.20 TUnZipper

70.20.1 Description

Extracts and decompresses files and directories in a .ZIP archive file.

70.20.2 Method overview

Page	Method	Description
1673	Clear	Removes all entries and files from object.
1671	Create	Constructor for the class instance.
1671	Destroy	Destructor for the class instance.
1673	Examine	Opens zip file and reads the directory entries (list of zipped files).
1673	Terminate	Sets the value in Terminated to True.
1672	Unzip	Unzips the specified .ZIP archive file.
1671	UnZipAllFiles	Unzips all files in a zip file, writing them to disk.
1672	UnZipFile	Unzips a single file found in the specified .ZIP archive.
1672	UnZipFiles	Unzips the specified files in a .ZIP archive file.
1671	UnZipOneFile	

70.20.3 Property overview

Page	Properties	Access	Description
1673	BufferSize	rw	Size of the buffer used to read and decompress entries in the .ZIP file.
1676	Entries	r	Collection with TFullZipFileEntry instances for files and directories stored in the .ZIP archive.
1676	FileComment	r	Comment stored in the .ZIP archive file.
1675	FileName	rw	Path and file name for the .zip file to be unzipped / processed.
1676	Files	r	Files in the zip file (deprecated).
1677	Flat	rw	Extracts files to a single directory.
1674	OnCloseInputStream	rw	Event handler signalled when the input stream for the .ZIP file is closed.
1674	OnCreateStream	rw	Event handler signalled when an output stream is created.
1674	OnDoneStream	rw	Event handler signalled when an output stream is closed.
1675	OnEndFile	rw	Callback procedure that will be called after unzipping a file.
1673	OnOpenInputStream	rw	Event handler signalled when the input stream for the .ZIP file is opened.
1674	OnPercent	rw	Threshold percentage which triggers a progress notification.
1674	OnProgress	rw	Progress event handler used when decompressing files.
1675	OnProgressEx	rw	Extended progress event handler used when decompressing files.
1675	OnStartFile	rw	Callback procedure that will be called before unzipping a file.
1675	OutputPath	rw	Path where archive files will be unzipped.
1677	Terminated	r	True if the Terminate method has been called.
1676	UseUTF8	rw	Indicates that the UTF-8-encoded names are used when locating and unzipping entries in the archive.

70.20.4 TUnZipper.Create

Synopsis: Constructor for the class instance.

Declaration:
```
constructor Create
```

Visibility: public

Description: Constructor for the class instance.

70.20.5 TUnZipper.Destroy

Synopsis: Destructor for the class instance.

Declaration:
```
destructor Destroy;  Override
```

Visibility: public

Description: Destructor for the class instance.

70.20.6 TUnZipper.UnZipOneFile

Declaration:
```
procedure UnZipOneFile(Item: TFullZipFileEntry);  Virtual
```

Visibility: public

70.20.7 TUnZipper.UnZipAllFiles

Synopsis: Unzips all files in a zip file, writing them to disk.

Declaration:
```
procedure UnZipAllFiles;  Virtual
procedure UnZipAllFiles(const AZipFileName: RawByteString)
```

Visibility: public

Description: This procedure unzips all files in a TZipper (1682) object and writes the unzipped files to disk.

The example below unzips the files into "C:\windows\temp":

```
uses
  Zipper;
var
  UnZipper: TUnZipper;
begin
  UnZipper := TUnZipper.Create;
  try
    UnZipper.FileName := ZipFilePath;
    UnZipper.OutputPath := 'C:\Windows\Temp';
    UnZipper.UnZipAllFiles;
  finally
    UnZipper.Free;
  end;
end.
```

70.20.8 TUnZipper.UnZipFile

Synopsis: Unzips a single file found in the specified .ZIP archive.

Declaration:
```
procedure UnZipFile(const aExtractFileName: RawByteString)
procedure UnZipFile(const AZipFileName: RawByteString;
                    const aExtractFileName: RawByteString)
```

Visibility: public

Description: Unzips a single file found in the specified .ZIP archive.

70.20.9 TUnZipper.UnZipFiles

Synopsis: Unzips the specified files in a .ZIP archive file.

Declaration:
```
procedure UnZipFiles(const AZipFileName: RawByteString;
                     FileList: TStrings)
procedure UnZipFiles(const AZipFileName: RawByteString;
                     aFileList: Array of RawBytestring)
procedure UnZipFiles(aFileList: TStrings)
```

Visibility: public

Description: Unzips the specified files in a .ZIP archive file.

70.20.10 TUnZipper.Unzip

Synopsis: Unzips the specified .ZIP archive file.

Declaration:
```
class procedure Unzip(const AZipFileName: RawByteString)
class procedure Unzip(const AZipFileName: RawByteString;
                      aExtractFileName: RawByteString)
class procedure UnZip(const AZipFileName: RawByteString;
                      const aExtractFileName: RawByteString;
                      aOutputFileName: AnsiString)
class procedure Unzip(const AZipFileName: RawByteString;
                      aFileList: Array of RawByteString)
class procedure Unzip(const AZipFileName: RawByteString;
                      aFileList: TStrings)
class procedure Unzip(const AZipFileName: RawByteString;
                      aFileList: Array of RawByteString;
                      aOutputDir: RawByteString; aFlat: Boolean)
class procedure Unzip(const AZipFileName: RawByteString;
                      aFileList: TStrings; aOutputDir: RawByteString;
                      aFlat: Boolean)
```

Visibility: public

Description: `UnZip` is an overloaded class method used to unzip one or more files in the specified .ZIP archive file. Overloaded variants are provided which allow the file or files to be specified using `RawByteString`, `Array`, or `TStrings` data types.

UnZip is a convenience method, and does not require an instance of the class. It uses the default options to perform the unzip operation.

70.20.11 TUnZipper.Clear

Synopsis: Removes all entries and files from object.

Declaration: `procedure Clear`

Visibility: public

Description: Removes all entries and files from object.

70.20.12 TUnZipper.Examine

Synopsis: Opens zip file and reads the directory entries (list of zipped files).

Declaration: `procedure Examine`

Visibility: public

Description: Opens zip file and reads the directory entries (list of zipped files).

70.20.13 TUnZipper.Terminate

Synopsis: Sets the value in Terminated to True.

Declaration: `procedure Terminate`

Visibility: public

Description: Sets the value in Terminated to True.

70.20.14 TUnZipper.BufferSize

Synopsis: Size of the buffer used to read and decompress entries in the .ZIP file.

Declaration: `Property BufferSize : LongWord`

Visibility: public

Access: Read,Write

Description: Size of the buffer used to read and decompress entries in the .ZIP file.

70.20.15 TUnZipper.OnOpenInputStream

Synopsis: Event handler signalled when the input stream for the .ZIP file is opened.

Declaration: `Property OnOpenInputStream : TCustomInputStreamEvent`

Visibility: public

Access: Read,Write

Description: Event handler signalled when the input stream for the .ZIP file is opened.

70.20.16 TUnZipper.OnCloseInputStream

Synopsis: Event handler signalled when the input stream for the .ZIP file is closed.

Declaration: `Property OnCloseInputStream : TCustomInputStreamEvent`

Visibility: public

Access: Read,Write

Description: Event handler signalled when the input stream for the .ZIP file is closed.

70.20.17 TUnZipper.OnCreateStream

Synopsis: Event handler signalled when an output stream is created.

Declaration: `Property OnCreateStream : TOnCustomStreamEvent`

Visibility: public

Access: Read,Write

Description: Event handler signalled when an output stream is created.

70.20.18 TUnZipper.OnDoneStream

Synopsis: Event handler signalled when an output stream is closed.

Declaration: `Property OnDoneStream : TOnCustomStreamEvent`

Visibility: public

Access: Read,Write

Description: Event handler signalled when an output stream is closed.

70.20.19 TUnZipper.OnPercent

Synopsis: Threshold percentage which triggers a progress notification.

Declaration: `Property OnPercent : Integer`

Visibility: public

Access: Read,Write

Description: Threshold percentage which triggers a progress notification.

70.20.20 TUnZipper.OnProgress

Synopsis: Progress event handler used when decompressing files.

Declaration: `Property OnProgress : TProgressEvent`

Visibility: public

Access: Read,Write

Description: Progress event handler used when decompressing files.

70.20.21 TUnZipper.OnProgressEx

Synopsis: Extended progress event handler used when decompressing files.

Declaration: `Property OnProgressEx : TProgressEventEx`

Visibility: public

Access: Read,Write

Description: Extended progress event handler used when decompressing files.

70.20.22 TUnZipper.OnStartFile

Synopsis: Callback procedure that will be called before unzipping a file.

Declaration: `Property OnStartFile : TOnStartFileEvent`

Visibility: public

Access: Read,Write

Description: Callback procedure that will be called before unzipping a file.

70.20.23 TUnZipper.OnEndFile

Synopsis: Callback procedure that will be called after unzipping a file.

Declaration: `Property OnEndFile : TOnEndOfFileEvent`

Visibility: public

Access: Read,Write

Description: Callback procedure that will be called after unzipping a file.

70.20.24 TUnZipper.FileName

Synopsis: Path and file name for the .zip file to be unzipped / processed.

Declaration: `Property FileName : RawByteString`

Visibility: public

Access: Read,Write

Description: Path and file name for the .zip file to be unzipped / processed.

70.20.25 TUnZipper.OutputPath

Synopsis: Path where archive files will be unzipped.

Declaration: `Property OutputPath : RawByteString`

Visibility: public

Access: Read,Write

Description: Path where archive files will be unzipped.

70.20.26 TUnZipper.FileComment

Synopsis: Comment stored in the .ZIP archive file.

Declaration: `Property FileComment : AnsiString`

Visibility: public

Access: Read

Description: Comment stored in the .ZIP archive file.

70.20.27 TUnZipper.Files

Synopsis: Files in the zip file (deprecated).

Declaration: `Property Files : TStrings`

Visibility: public

Access: Read

Description: List of files that should be compressed in the zip file. Deprecated. Use Entries.AddFileEntry(FileName) or Entries.AddFileEntries(List) instead.

70.20.28 TUnZipper.Entries

Synopsis: Collection with TFullZipFileEntry instances for files and directories stored in the .ZIP archive.

Declaration: `Property Entries : TFullZipFileEntries`

Visibility: public

Access: Read

Description: `Entries` is a read-only `TFullZipFileEntries` property, and the collection representing the items stored in the .ZIP archive file. Entries contains `TFullZipFileEntry` instances which represent the files or directories present in the .ZIP file.

Values in the Entries collection are created and stored when file directory in the .ZIP file is read. This can occur when the `Examine` method is called, or when extracting one or more files using the `UnZipAllFiles` / `UnZipFiles` / `UnZipFile` methods.

The items in the Entries collection are removed when the `Clear` method is called.

See also: TUnZipper.Examine (1673), TUnZipper.Clear (1673), TUnZipper.UnZipAllFiles (1671), TUnZipper.UnZipFiles (1672), TUnZipper.UnZipFile (1672), TUnZipper.UnZip (1672), TFullZipFileEntries (1666), TFullZipFileEntry (1666)

70.20.29 TUnZipper.UseUTF8

Synopsis: Indicates that the UTF-8-encoded names are used when locating and unzipping entries in the archive.

Declaration: `Property UseUTF8 : Boolean`

Visibility: public

Access: Read,Write

Description: `UseUTF8` is a `Boolean` property which indicates if UTF-8-encoded names are used when locating and unzipping items stored in the .ZIP archive.

Each `TZipFileEntry` instance stored in the `Entries` collection has both UTF-8-encoded and RawByteString (same as AnsiString with no code page) variants of file or directory names. Set UseUTF8 to **True** to use the UTF-8-encoded version. The default value is **False**, and causes the `RawByteString` version to be used.

UseUTF8 is used when methods like `UnZipAllFiles`, `UnZipFiles`, and `UnZipFile` are called.

See also: TUnZipper.Entries (1676), TUnZipper.UnZipAllFiles (1671), TUnZipper.UnZipFiles (1672), TUnZipper.UnZipFile (1672), TZipFileEntries (1677), TZipFileEntry (1679)

70.20.30 TUnZipper.Flat

Synopsis: Extracts files to a single directory.

Declaration: `Property Flat : Boolean`

Visibility: public

Access: Read,Write

Description: Enables flat extraction; like -j (also called junk paths) when using the unzip command-line utility. Directory structure(s) in the .zip file are not recreated, and files are extracted to the same directory.

70.20.31 TUnZipper.Terminated

Synopsis: True if the Terminate method has been called.

Declaration: `Property Terminated : Boolean`

Visibility: public

Access: Read

Description: True if the Terminate method has been called.

70.21 TZipFileEntries

70.21.1 Description

`TZipFileEntries` is a `TCollection` descendant which provides support for using `TZipFileEntry` instances as the Items in the collection. It provides an indexed `Entries` property used to access the TZipFileEntry instances in the collection, and serves as the default property for enumerator access.

TZipFileEntries is the type used to implement the `Entries` property in `TZipper`.

See also: TZipFileEntries.Entries (1678), TZipFileEntry (1679), TZipper.Entries (1688), TUnZipper.Entries (1676), TFullZipFileEntries (1666)

70.21.2 Method overview

Page	Method	Description
1678	AddFileEntries	Adds TZipFileEntry instances in the collection for the file names in List.
1678	AddFileEntry	Adds file to zip directory.

70.21.3 Property overview

Page	Properties	Access	Description
1678	Entries	rw	Entries (files) in the zip archive.

70.21.4 TZipFileEntries.AddFileEntry

Synopsis: Adds file to zip directory.

Declaration:
```
function AddFileEntry(const ADiskFileName: AnsiString) : TZipFileEntry
function AddFileEntry(const ADiskFileName: AnsiString;
                      const AArchiveFileName: AnsiString) : TZipFileEntry
function AddFileEntry(const AStream: TStream;
                      const AArchiveFileName: AnsiString) : TZipFileEntry
```

Visibility: public

Description: `AddFileEntry` adds a file or directory to the list of entries that will be written out in the .zip file. AddFileEntry calls the `Add` method to create the new collection item, and casts it the `TZipFileEntry` type used in `TZipFileEntries`.

Values passed as arguments to the overloaded variants are stored in the corresponding properties in the TZipFileEntry instance.

The return value is the TZipFileEntry instance added to the collection.

70.21.5 TZipFileEntries.AddFileEntries

Synopsis: Adds TZipFileEntry instances in the collection for the file names in List.

Declaration:
```
procedure AddFileEntries(const List: TStrings)
```

Visibility: public

Description: `AddFileEntries` is a method used to add a list of files names to the collection. `List` contains the file names added in the method. AddFileEntries iterates over the string values in List, and calls the `AddFileEntry` method to create new items in the collection.

See also: TZipFileEntries.AddFileEntry (1678)

70.21.6 TZipFileEntries.Entries

Synopsis: Entries (files) in the zip archive.

Declaration:
```
Property Entries[AIndex: Integer]: TZipFileEntry; default
```

Visibility: public

Access: Read,Write

Description: `Entries` is an indexed `TZipFileEntry` property which provides indexed access to the `Items` in the collection by their ordinal position. The item values are cast to the `TZipFileEntry` type used in `TZipFileEntries`.

Entries is the default property in TZipFileEntries, and allows an enumerator to be used to access the TZipFileEntry values in the collection.

See also: TZipFileEntry (1679), TCollection.Items (**??**)

70.22 TZipFileEntry

70.22.1 Description

`TZipFileEntry` is a `TCollectionItem` descendant which represents a file or directory added to a .ZIP file archive. TZipFileEntry is the type used for items in the `Entries` property in the `TZipFileEntries` collection.

TZipFileEntry provides properties with metadata for the file or directory, including:

ArchiveFileName Name of the file or directory in the .ZIP archive.

UTF8ArchiveFileName Name of the file or directory in the .ZIP archive using UTF-8 encoding.

DiskFileName Name of the file or directory on the local file system.

UTF8DiskFileName Name of the file or directory using UTF-8 encoding.

Size Size of the compressed file or directory in the .ZIP archive.

DateTime The timestamp for file or directory in the .ZIP archive.

OS Indicates the operating system device type / file system where the file or directory originated.

Attributes File attributes for the entry.

CompressionLevel Compression level applied to the content in the .ZIP archive.

Stream TStream instance with the content for the entry.

Use `IsDirectory` to determine if the entry represents a directory.

Use `IsLink` to determine if the entry is a symbolic link on the local file system.

70.22.2 Method overview

Page	Method	Description
1680	Assign	Copies property values from the specified persistent object.
1680	Create	Constructor for the class instance.
1680	IsDirectory	True if the entry is a directory on the local file system.
1680	IsLink	True if the directory is a symbolic link on the local file system.

70.22.3 Property overview

Page	Properties	Access	Description
1681	ArchiveFileName	rw	Name of the file or directory in the .ZIP archive.
1682	Attributes	rw	File attributes for the file or directory.
1682	CompressionLevel	rw	Compression level applied to the content stored in the .ZIP archive.
1682	DateTime	rw	Timestamp for the file or directory in the .ZIP archive.
1681	DiskFileName	rw	Name of the file or directory on the local file system.
1682	OS	rw	Indication of operating system/file system.
1681	Size	rw	Size of the compressed content for the file or directory.
1680	Stream	rw	Stream with the content for the entry.
1681	UTF8ArchiveFileName	rw	Archive filename as UTF8 string.
1681	UTF8DiskFileName	rw	Name of the file or directory on the local file system using UTF-8 encoding.

70.22.4 TZipFileEntry.Create

Synopsis: Constructor for the class instance.

Declaration: `constructor Create(ACollection: TCollection); Override`

Visibility: public

Description: `Create` is the overridden constructor for the class instance. Create sets the default values for properties, including:

DateTime Sets to the current date and time for the local computer.

OS Set to OS_UNIX for UNIX-like environments, or OS_VFAT for all others.

Attributes Set to 0 (no attributes).

CompressionLevel Set to clDefault.

Create calls the inherited constructor prior to exiting from the method.

70.22.5 TZipFileEntry.IsDirectory

Synopsis: True if the entry is a directory on the local file system.

Declaration: `function IsDirectory : Boolean`

Visibility: public

Description: True if the entry is a directory on the local file system.

70.22.6 TZipFileEntry.IsLink

Synopsis: True if the directory is a symbolic link on the local file system.

Declaration: `function IsLink : Boolean`

Visibility: public

Description: True if the directory is a symbolic link on the local file system.

70.22.7 TZipFileEntry.Assign

Synopsis: Copies property values from the specified persistent object.

Declaration: `procedure Assign(Source: TPersistent); Override`

Visibility: public

Description: Copies property values from the specified persistent object.

70.22.8 TZipFileEntry.Stream

Synopsis: Stream with the content for the entry.

Declaration: `Property Stream : TStream`

Visibility: public

Access: Read,Write

Description: Stream with the content for the entry.

70.22.9 TZipFileEntry.ArchiveFileName

Synopsis: Name of the file or directory in the .ZIP archive.

Declaration: `Property ArchiveFileName : AnsiString`

Visibility: published

Access: Read,Write

Description: Name of the file or directory in the .ZIP archive.

70.22.10 TZipFileEntry.UTF8ArchiveFileName

Synopsis: Archive filename as UTF8 string.

Declaration: `Property UTF8ArchiveFileName : UTF8String`

Visibility: published

Access: Read,Write

Description: `UTF8ArchiveFileName` is the filename in UTF8-format. Use this if you need to have filenames with characters not in ASCII range.

70.22.11 TZipFileEntry.DiskFileName

Synopsis: Name of the file or directory on the local file system.

Declaration: `Property DiskFileName : AnsiString`

Visibility: published

Access: Read,Write

Description: Name of the file or directory on the local file system.

70.22.12 TZipFileEntry.UTF8DiskFileName

Synopsis: Name of the file or directory on the local file system using UTF-8 encoding.

Declaration: `Property UTF8DiskFileName : UTF8String`

Visibility: published

Access: Read,Write

Description: Name of the file or directory on the local file system using UTF-8 encoding.

70.22.13 TZipFileEntry.Size

Synopsis: Size of the compressed content for the file or directory.

Declaration: `Property Size : Int64`

Visibility: published

Access: Read,Write

Description: Size of the compressed content for the file or directory.

70.22.14 TZipFileEntry.DateTime

Synopsis: Timestamp for the file or directory in the .ZIP archive.

Declaration: `Property DateTime : TDateTime`

Visibility: published

Access: Read,Write

Description: Timestamp for the file or directory in the .ZIP archive.

70.22.15 TZipFileEntry.OS

Synopsis: Indication of operating system/file system.

Declaration: `Property OS : Byte`

Visibility: published

Access: Read,Write

Description: Currently either OS_UNIX (if UNIX is defined) or OS_FAT.

70.22.16 TZipFileEntry.Attributes

Synopsis: File attributes for the file or directory.

Declaration: `Property Attributes : LongWord`

Visibility: published

Access: Read,Write

Description: File attributes for the file or directory.

70.22.17 TZipFileEntry.CompressionLevel

Synopsis: Compression level applied to the content stored in the .ZIP archive.

Declaration: `Property CompressionLevel : Tcompressionlevel`

Visibility: published

Access: Read,Write

Description: Compression level applied to the content stored in the .ZIP archive.

70.23 TZipper

70.23.1 Description

Creates a .ZIP archive file.

70.23.2 Method overview

Page	Method	Description
1686	Clear	Removes all values in the Entries and Files properties.
1683	Create	Constructor for the class instance.
1683	Destroy	Destructor for the class instance.
1684	SaveToFile	Saves the archive to a file with a new name.
1684	SaveToStream	Save the archive to a stream.
1686	Terminate	Halts an assigned compressor in the class instance, and sets Terminated to True.
1685	Zip	Convenience method used to create a .zip file with the given name containing the specified file(s).
1684	ZipAllFiles	Zips all files in object and writes zip to disk.
1685	ZipFile	Zip one file to a zip file.
1685	ZipFiles	Zip multiple files into an archive.

70.23.3 Property overview

Page	Properties	Access	Description
1686	BufferSize	rw	Buffer size used when reading and processing files.
1688	Entries	rw	Collection with the TZipFileEntry instances in the .ZIP archive.
1687	FileComment	rw	Comment stored in the .ZIP archive file.
1687	FileName	rw	Name of the .ZIP archive file where the compressed files and directories are stored.
1688	Files	r	Provides access to the list of files and directories in the archive.
1688	InMemSize	rw	Total memory used for the compressed content in the .ZIP file.
1687	OnEndFile	rw	Event handler signalled when compression for a file has been completed.
1686	OnPercent	rw	Threshold percentage which triggers progress notifications when processing files.
1687	OnProgress	rw	Event handler signalled to show a percent complete progress notifications.
1687	OnStartFile	rw	Event handler signalled when compression for a file is started.
1688	Terminated	r	True if the Terminate method has been called.
1689	UseLanguageEncoding	rw	Use language encoding.

70.23.4 TZipper.Create

Synopsis: Constructor for the class instance.

Declaration: `constructor Create`

Visibility: public

Description: Constructor for the class instance.

70.23.5 TZipper.Destroy

Synopsis: Destructor for the class instance.

Declaration: `destructor Destroy;  Override`

Visibility: public

Description: Destructor for the class instance.

70.23.6 TZipper.ZipAllFiles

Synopsis: Zips all files in object and writes zip to disk.

Declaration: `procedure ZipAllFiles;  Virtual`

Visibility: public

Description: This procedure zips up all files in the TZipper (1682) object and writes the resulting zip file to disk.

An example of using this procedure:

```
uses
  Zipper;
var
  Zipper: TZipper;
begin
  try
    Zipper := TZipper.Create;
    Zipper.FileName := ParamStr(1); //Use the first parameter on the command line as
    for I := 2 to ParamCount do //Use the other arguments on the command line as fil
      Zipper.Entries.AddFileEntry(ParamStr(I), ParamStr(I));
    Zipper.ZipAllFiles;
  finally
    Zipper.Free;
  end;
end.
```

70.23.7 TZipper.SaveToFile

Synopsis: Saves the archive to a file with a new name.

Declaration: `procedure SaveToFile(const AFileName: RawByteString)`

Visibility: public

Description: Saves a .ZIP file with a new name.

70.23.8 TZipper.SaveToStream

Synopsis: Save the archive to a stream.

Declaration: `procedure SaveToStream(AStream: TStream)`

Visibility: public

Description: Save the archive to a stream.

70.23.9 TZipper.ZipFile

Synopsis: Zip one file to a zip file.

Declaration:
```
procedure ZipFile(const aFileToBeZipped: RawByteString)
procedure ZipFile(const AZipFileName: RawByteString;
                  const aFileToBeZipped: RawByteString)
```

Visibility: public

Description: Zips the specified files into a zip with the name in AFileName.

See also: ZipFiles (1685)

70.23.10 TZipper.ZipFiles

Synopsis: Zip multiple files into an archive.

Declaration:
```
procedure ZipFiles(const AZipFileName: RawByteString;
                   FileList: TStrings)
procedure ZipFiles(const AZipFileName: RawByteString;
                   const FileList: Array of RawbyteString)
procedure ZipFiles(const aFileList: Array of RawbyteString)
procedure ZipFiles(FileList: TStrings)
procedure ZipFiles(const AZipFileName: RawByteString;
                   Entries: TZipFileEntries)
procedure ZipFiles(Entries: TZipFileEntries)
```

Visibility: public

Description: Zip multiple files into an archive.

See also: ZipFile (1685)

70.23.11 TZipper.Zip

Synopsis: Convenience method used to create a .zip file with the given name containing the specified file(s).

Declaration:
```
class procedure Zip(const AZipFileName: RawByteString;
                    const aFileToBeZipped: RawByteString)
class procedure Zip(const AZipFileName: RawByteString;
                    aFileList: Array of RawByteString)
class procedure Zip(const AZipFileName: RawByteString;
                    aFileList: TStrings)
```

Visibility: public

Description: Zip is a class procedure used to create a .zip file with the name specified in AZipFileName. Overloaded variants are provided that allow one or more file names to be specified using the `AFileToBeZipped` or `AFileList` arguments.

For example:

```
var
  SZip, SFile: RawByteString;

  SZip := '/usr/tmp/docbook5-catalog.zip';
  SFile := '/usr/share/xml/docbook/schema/sch/5.0/catalog.xml'
```

```
TZipper.Zip(SZip, SFile);
```

See also: TUnzipper.Unzip (1672)

70.23.12 TZipper.Clear

Synopsis: Removes all values in the Entries and Files properties.

Declaration: `procedure Clear`

Visibility: public

Description: Removes all values in the Entries and Files properties.

70.23.13 TZipper.Terminate

Synopsis: Halts an assigned compressor in the class instance, and sets Terminated to True.

Declaration: `procedure Terminate`

Visibility: public

Description: Halts an assigned compressor in the class instance, and sets Terminated to True.

70.23.14 TZipper.BufferSize

Synopsis: Buffer size used when reading and processing files.

Declaration: `Property BufferSize : LongWord`

Visibility: public

Access: Read,Write

Description: Buffer size used when reading and processing files.

70.23.15 TZipper.OnPercent

Synopsis: Threshold percentage which triggers progress notifications when processing files.

Declaration: `Property OnPercent : Integer`

Visibility: public

Access: Read,Write

Description: Threshold percentage which triggers progress notifications when processing files.

70.23.16 TZipper.OnProgress

Synopsis: Event handler signalled to show a percent complete progress notifications.

Declaration: `Property OnProgress : TProgressEvent`

Visibility: public

Access: Read,Write

Description: Event handler signalled to show a percent complete progress notifications.

70.23.17 TZipper.OnStartFile

Synopsis: Event handler signalled when compression for a file is started.

Declaration: `Property OnStartFile : TOnStartFileEvent`

Visibility: public

Access: Read,Write

Description: Event handler signalled when compression for a file is started.

70.23.18 TZipper.OnEndFile

Synopsis: Event handler signalled when compression for a file has been completed.

Declaration: `Property OnEndFile : TOnEndOfFileEvent`

Visibility: public

Access: Read,Write

Description: Event handler signalled when compression for a file has been completed.

70.23.19 TZipper.FileName

Synopsis: Name of the .ZIP archive file where the compressed files and directories are stored.

Declaration: `Property FileName : RawByteString`

Visibility: public

Access: Read,Write

Description: Name of the .ZIP archive file where the compressed files and directories are stored.

70.23.20 TZipper.FileComment

Synopsis: Comment stored in the .ZIP archive file.

Declaration: `Property FileComment : AnsiString`

Visibility: public

Access: Read,Write

Description: Comment stored in the .ZIP archive file.

70.23.21 TZipper.Files

Synopsis: Provides access to the list of files and directories in the archive.

Declaration: `Property Files : TStrings; deprecated;`

Visibility: public

Access: Read

Description: Deprecated. Use the Entries property to call its `AddFileEntry` or `AddFileEntries` methods instead.

See also: TZipper.Entries (1688), TZipperFileEntries.AddFileEntry (1651), TZipperFileEntries.AddFileEntries (1651)

70.23.22 TZipper.InMemSize

Synopsis: Total memory used for the compressed content in the .ZIP file.

Declaration: `Property InMemSize : Int64`

Visibility: public

Access: Read,Write

Description: Total memory used for the compressed content in the .ZIP file.

70.23.23 TZipper.Entries

Synopsis: Collection with the TZipFileEntry instances in the .ZIP archive.

Declaration: `Property Entries : TZipFileEntries`

Visibility: public

Access: Read,Write

Description: Collection with the TZipFileEntry instances in the .ZIP archive.

70.23.24 TZipper.Terminated

Synopsis: True if the Terminate method has been called.

Declaration: `Property Terminated : Boolean`

Visibility: public

Access: Read

Description: True if the Terminate method has been called.

70.23.25 TZipper.UseLanguageEncoding

Synopsis: Use language encoding.

Declaration: `Property UseLanguageEncoding : Boolean`

Visibility: public

Access: Read,Write

Description: EFS/language encoding using UTF-8.

Chapter 71

Reference for unit 'ZStream'

71.1 Used units

Table 71.1: Used units by unit 'ZStream'

Name	Page
Classes	**??**
gzio	**??**
System	**??**
zbase	**??**

71.2 Overview

The ZStream unit implements a TStream (**??**) descendent (TCompressionStream (1691)) which uses the deflate algorithm to compress everything that is written to it. The compressed data is written to the output stream, which is specified when the compressor class is created.

Likewise, a `TStream` descendent is implemented which reads data from an input stream (TDecompressionStream (1694)) and decompresses it with the inflate algorithm.

71.3 Constants, types and variables

71.3.1 Types

```
Tcompressionlevel = (clnone,clfastest,cldefault,clmax)
```

Table 71.2: Enumeration values for type Tcompressionlevel

Value	Explanation
cldefault	Use default compression.
clfastest	Use fast (but less) compression.
clmax	Use maximum compression.
clnone	Do not use compression, just copy data.

Compression level for the deflate algorithm.

```
Tgzopenmode = (gzopenread,gzopenwrite)
```

Table 71.3: Enumeration values for type Tgzopenmode

Value	Explanation
gzopenread	Open file for reading.
gzopenwrite	Open file for writing.

Open mode for gzip file.

71.4 Ecompressionerror

71.4.1 Description

`ECompressionError` is the exception class used by the TCompressionStream (1691) class.

71.5 Edecompressionerror

71.5.1 Description

`EDecompressionError` is the exception class used by the TDeCompressionStream (1694) class.

71.6 Egzfileerror

71.6.1 Description

`Egzfileerror` is the exception class used to report errors by the Tgzfilestream (1697) class.

See also: Tgzfilestream (1697)

71.7 Ezliberror

71.7.1 Description

Errors which occur in the zstream unit are signaled by raising an `EZLibError` exception descendent.

71.8 Tcompressionstream

71.8.1 Description

`TCompressionStream`

71.8.2 Method overview

Page	Method	Description
1692	create	Create a new instance of the compression stream.
1692	destroy	Flushes data to the output stream and destroys the compression stream.
1693	flush	Flush remaining data to the target stream.
1693	get_compressionrate	Get the current compression rate.
1692	write	Write data to the stream.

71.8.3 Property overview

Page	Properties	Access	Description
1693	OnProgress		Progress handler.

71.8.4 Tcompressionstream.create

Synopsis: Create a new instance of the compression stream.

Declaration:
```
constructor create(level: Tcompressionlevel; dest: TStream;
                   Askipheader: Boolean)
```

Visibility: public

Description: `Create` creates a new instance of the compression stream. It merely calls the inherited constructor with the destination stream `Dest` and stores the compression level.

If `ASkipHeader` is set to `True`, the method will not write the block header to the stream. This is required for deflated data in a zip file.

Note that the compressed data is only completely written after the compression stream is destroyed.

See also: Destroy (1692)

71.8.5 Tcompressionstream.destroy

Synopsis: Flushes data to the output stream and destroys the compression stream.

Declaration:
```
destructor destroy;  Override
```

Visibility: public

Description: `Destroy` flushes the output stream: any compressed data not yet written to the output stream are written, and the deflate structures are cleaned up.

Errors: None.

See also: Create (1692)

71.8.6 Tcompressionstream.write

Synopsis: Write data to the stream.

Declaration:
```
function write(const buffer; count: LongInt) : LongInt;  Override
```

Visibility: public

Description: `Write` takes `Count` bytes from `Buffer` and compresses (deflates) them. The compressed result is written to the output stream.

Errors: If an error occurs, an ECompressionError (1691) exception is raised.

See also: Write (1692), ECompressionError (1691)

71.8.7 Tcompressionstream.flush

Synopsis: Flush remaining data to the target stream.

Declaration: `procedure flush`

Visibility: public

Description: `flush` writes any remaining data in the memory buffers to the target stream, and clears the memory buffer.

71.8.8 Tcompressionstream.get_compressionrate

Synopsis: Get the current compression rate.

Declaration: `function get_compressionrate : single`

Visibility: public

Description: `get_compressionrate` returns the percentage of the number of written compressed bytes relative to the number of written bytes.

Errors: If no bytes were written, an exception is raised.

71.8.9 Tcompressionstream.OnProgress

Synopsis: Progress handler.

Declaration: `Property OnProgress :`

Visibility: public

Access:

Description: `OnProgress` is called whenever output data is written to the output stream. It can be used to update a progress bar or so. The `Sender` argument to the progress handler is the compression stream instance.

71.9 Tcustomzlibstream

71.9.1 Description

`TCustomZlibStream` serves as the ancestor class for the TCompressionStream (1691) and TDecompressionStream (1694) classes.

It introduces support for a progress handler, and stores the input or output stream.

71.9.2 Method overview

Page	Method	Description
1694	create	Create a new instance of `TCustomZlibStream`.
1694	destroy	Clear up instance.

71.9.3 Tcustomzlibstream.create

Synopsis: Create a new instance of `TCustomZlibStream`.

Declaration: `constructor create(stream: TStream)`

Visibility: public

Description: `Create` creates a new instance of `TCustomZlibStream`. It stores a reference to the input/output stream, and initializes the deflate compression mechanism so they can be used by the descendents.

See also: TCompressionStream (1691), TDecompressionStream (1694)

71.9.4 Tcustomzlibstream.destroy

Synopsis: Clear up instance.

Declaration: `destructor destroy;  Override`

Visibility: public

Description: `Destroy` cleans up the internal memory buffer and calls the inherited destroy.

See also: Tcustomzlibstream.create (1694)

71.10 Tdecompressionstream

71.10.1 Description

`TDecompressionStream` performs the inverse operation of TCompressionStream (1691). A read operation reads data from an input stream and decompresses (inflates) the data it as it goes along.

The decompression stream reads it's compressed data from a stream with deflated data. This data can be created e.g. with a TCompressionStream (1691) compression stream.

See also: TCompressionStream (1691)

71.10.2 Method overview

Page	Method	Description
1695	create	Creates a new instance of the `TDecompressionStream` stream.
1695	destroy	Destroys the `TDecompressionStream` instance.
1696	get_compressionrate	Get the current compression rate.
1695	read	Read data from the compressed stream.
1696	Seek	Move stream position to a certain location in the stream.

71.10.3 Property overview

Page	Properties	Access	Description
1696	OnProgress		Progress handler.

71.10.4 Tdecompressionstream.create

Synopsis: Creates a new instance of the `TDecompressionStream` stream.

Declaration:
```
constructor create(Asource: TStream; Askipheader: Boolean)
```

Visibility: public

Description: `Create` creates and initializes a new instance of the `TDecompressionStream` class. It calls the inherited `Create` and passes it the `Source` stream. The source stream is the stream from which the compressed (deflated) data is read.

If `ASkipHeader` is true, then the gzip data header is skipped, allowing `TDecompressionStream` to read deflated data in a .zip file. (this data does not have the gzip header record prepended to it).

Note that the source stream is by default not owned by the decompression stream, and is not freed when the decompression stream is destroyed.

See also: Destroy (1695)

71.10.5 Tdecompressionstream.destroy

Synopsis: Destroys the `TDecompressionStream` instance.

Declaration:
```
destructor destroy;  Override
```

Visibility: public

Description: `Destroy` cleans up the inflate structure, and then simply calls the inherited destroy.

By default the source stream is not freed when calling `Destroy`.

See also: Create (1695)

71.10.6 Tdecompressionstream.read

Synopsis: Read data from the compressed stream.

Declaration:
```
function read(var buffer; count: LongInt) : LongInt;  Override
```

Visibility: public

Description: `Read` will read data from the compressed stream until the decompressed data size is `Count` or there is no more compressed data available. The decompressed data is written in `Buffer`. The function returns the number of bytes written in the buffer.

Errors: If an error occurs, an EDeCompressionError (1691) exception is raised.

See also: Write (1692)

71.10.7 Tdecompressionstream.Seek

Synopsis: Move stream position to a certain location in the stream.

Declaration:
```
function Seek(const Offset: Int64; Origin: TSeekOrigin) : Int64
                  ;  Override
```

Visibility: public

Description: `Seek` overrides the standard `Seek` implementation. There are a few differences between the implementation of `Seek` in Free Pascal compared to Delphi:

- In Free Pascal, you can perform any seek. In case of a forward seek, the Free Pascal implementation will read some bytes until the desired position is reached, in case of a backward seek it will seek the source stream backwards to the position it had at the creation time of the `TDecompressionStream` and then again read some bytes until the desired position has been reached.
- In Free Pascal, a seek with soFromBeginning will reset the source stream to the position it had when the `TDecompressionStream` was created. In Delphi, the source stream is reset to position 0. This means that at creation time the source stream must always be at the start of the zstream, you cannot use `TDecompressionStream.Seek` to reset the source stream to the begin of the file.

Errors: An EDecompressionError (1691) exception is raised if the stream does not allow the requested seek operation.

See also: Read (1695)

71.10.8 Tdecompressionstream.get_compressionrate

Synopsis: Get the current compression rate.

Declaration:
```
function get_compressionrate : single
```

Visibility: public

Description: `get_compressionrate` returns the percentage of the number of read compressed bytes relative to the total number of read bytes.

Errors: If no bytes were written, an exception is raised.

71.10.9 Tdecompressionstream.OnProgress

Synopsis: Progress handler.

Declaration:
```
Property OnProgress :
```

Visibility: public

Access:

Description: `OnProgress` is called whenever input data is read from the source stream. It can be used to update a progress bar or so. The `Sender` argument to the progress handler is the decompression stream instance.

71.11 TGZFileStream

71.11.1 Description

`TGZFileStream` can be used to read data from a gzip file, or to write data to a gzip file.

See also: TCompressionStream (1691), TDeCompressionStream (1694)

71.11.2 Method overview

Page	Method	Description
1697	create	Create a new instance of `TGZFileStream`.
1698	destroy	Removes `TGZFileStream` instance.
1697	read	Read data from the compressed file.
1698	seek	Set the position in the compressed stream.
1698	write	Write data to be compressed.

71.11.3 TGZFileStream.create

Synopsis: Create a new instance of `TGZFileStream`.

Declaration:
```
constructor create(filename: ansistring; filemode: Tgzopenmode)
```

Visibility: public

Description: `Create` creates a new instance of the `TGZFileStream` class. It opens `FileName` for reading or writing, depending on the `FileMode` parameter. It is not possible to open the file read-write. If the file is opened for reading, it must exist.

If the file is opened for reading, the TGZFileStream.Read (1697) method can be used for reading the data in uncompressed form.

If the file is opened for writing, any data written using the TGZFileStream.Write (1698) method will be stored in the file in compressed (deflated) form.

Errors: If the file is not found, an EZlibError (1691) exception is raised.

See also: Destroy (1698), TGZOpenMode (1691)

71.11.4 TGZFileStream.read

Synopsis: Read data from the compressed file.

Declaration:
```
function read(var buffer; count: LongInt) : LongInt;  Override
```

Visibility: public

Description: `Read` overrides the `Read` method of `TStream` to read the data from the compressed file. The `Buffer` parameter indicates where the read data should be stored. The `Count` parameter specifies the number of bytes (*uncompressed*) that should be read from the compressed file. Note that it is not possible to read from the stream if it was opened in write mode.

The function returns the number of uncompressed bytes actually read.

Errors: If `Buffer` points to an invalid location, or does not have enough room for `Count` bytes, an exception will be raised.

See also: Create (1697), Write (1698), Seek (1698)

71.11.5 TGZFileStream.write

Synopsis: Write data to be compressed.

Declaration: `function write(const buffer; count: LongInt) : LongInt;  Override`

Visibility: public

Description: `Write` writes `Count` bytes from `Buffer` to the compressed file. The data is compressed as it is written, so ideally, less than `Count` bytes end up in the compressed file. Note that it is not possible to write to the stream if it was opened in read mode.

The function returns the number of (uncompressed) bytes that were actually written.

Errors: In case of an error, an EZlibError (1691) exception is raised.

See also: Create (1697), Read (1697), Seek (1698)

71.11.6 TGZFileStream.seek

Synopsis: Set the position in the compressed stream.

Declaration: `function seek(offset: LongInt; origin: Word) : LongInt;  Override`

Visibility: public

Description: `Seek` sets the position to `Offset` bytes, starting from `Origin`. Not all combinations are possible, see TDecompressionStream.Seek (1696) for a list of possibilities.

Errors: In case an impossible combination is asked, an EZlibError (1691) exception is raised.

See also: TDecompressionStream.Seek (1696)

71.11.7 TGZFileStream.destroy

Synopsis: Removes `TGZFileStream` instance.

Declaration: `destructor destroy;  Override`

Visibility: public

Description: `Destroy` closes the file and releases the `TGZFileStream` instance from memory.

See also: Create (1697)

71.12 TGZipCompressionStream

71.12.1 Method overview

Page	Method	Description
1698	Create	
1699	Destroy	
1699	Write	

71.12.2 TGZipCompressionStream.Create

Declaration:
```
constructor Create(ADest: TStream);  Overload
constructor Create(ALevel: Tcompressionlevel; ADest: TStream);  Overload
```

Visibility: public

71.12.3 TGZipCompressionStream.Destroy

Declaration: `destructor Destroy;  Override`

Visibility: public

71.12.4 TGZipCompressionStream.Write

Declaration: `function Write(const Buffer; Count: LongInt) : LongInt;  Override`

Visibility: public

71.13 TGZipDecompressionStream

71.13.1 Method overview

Page	Method	Description
1699	Create	
1699	Destroy	
1699	Read	
1699	Seek	

71.13.2 TGZipDecompressionStream.Create

Declaration: `constructor Create(ASource: TStream)`

Visibility: public

71.13.3 TGZipDecompressionStream.Destroy

Declaration: `destructor Destroy;  Override`

Visibility: public

71.13.4 TGZipDecompressionStream.Read

Declaration: `function Read(var Buffer; Count: LongInt) : LongInt;  Override`

Visibility: public

71.13.5 TGZipDecompressionStream.Seek

Declaration: `function Seek(const Offset: Int64; Origin: TSeekOrigin) : Int64 ;  Override`

Visibility: public