Free Component Library (FCL): Reference guide.

Reference guide for FCL units.
Document version 3.2.4
September 2026

Michaël Van Canneyt

Contents

0.1 Overview . . . 94

1 Reference for unit 'AdvancedSingleInstance' 95

1.1 Used units . . . 95
1.2 Overview . . . 95
1.3 Constants, types and variables . . . 95
1.3.1 Types . . . 95
1.4 TAdvancedSingleInstance . . . 96
1.4.1 Description . . . 96
1.4.2 Method overview . . . 96
1.4.3 Property overview . . . 96
1.4.4 TAdvancedSingleInstance.Create . . . 96
1.4.5 TAdvancedSingleInstance.Start . . . 97
1.4.6 TAdvancedSingleInstance.Stop . . . 97
1.4.7 TAdvancedSingleInstance.ServerCheckMessages . . . 97
1.4.8 TAdvancedSingleInstance.ClientPostParams . . . 97
1.4.9 TAdvancedSingleInstance.ClientPostCustomRequest . . . 98
1.4.10 TAdvancedSingleInstance.ClientSendCustomRequest . . . 98
1.4.11 TAdvancedSingleInstance.ServerPostCustomResponse . . . 98
1.4.12 TAdvancedSingleInstance.ClientPeekCustomResponse . . . 99
1.4.13 TAdvancedSingleInstance.ID . . . 99
1.4.14 TAdvancedSingleInstance.Global . . . 99
1.4.15 TAdvancedSingleInstance.OnServerReceivedCustomRequest . . . 99

2 Reference for unit 'ascii85' 100

2.1 Used units . . . 100
2.2 Overview . . . 100
2.3 Constants, types and variables . . . 100
2.3.1 Types . . . 100
2.4 TASCII85DecoderStream . . . 101
2.4.1 Description . . . 101
2.4.2 Method overview . . . 101

2.4.3 Property overview . . . 101
2.4.4 TASCII85DecoderStream.Create . . . 101
2.4.5 TASCII85DecoderStream.Decode . . . 102
2.4.6 TASCII85DecoderStream.Close . . . 102
2.4.7 TASCII85DecoderStream.ClosedP . . . 102
2.4.8 TASCII85DecoderStream.Destroy . . . 102
2.4.9 TASCII85DecoderStream.Read . . . 103
2.4.10 TASCII85DecoderStream.Seek . . . 103
2.4.11 TASCII85DecoderStream.BExpectBoundary . . . 103
2.5 TASCII85EncoderStream . . . 103
2.5.1 Description . . . 103
2.5.2 Method overview . . . 104
2.5.3 Property overview . . . 104
2.5.4 TASCII85EncoderStream.Create . . . 104
2.5.5 TASCII85EncoderStream.Destroy . . . 104
2.5.6 TASCII85EncoderStream.Write . . . 104
2.5.7 TASCII85EncoderStream.Width . . . 105
2.5.8 TASCII85EncoderStream.Boundary . . . 105
2.6 TASCII85RingBuffer . . . 105
2.6.1 Description . . . 105
2.6.2 Method overview . . . 105
2.6.3 Property overview . . . 105
2.6.4 TASCII85RingBuffer.Write . . . 106
2.6.5 TASCII85RingBuffer.Read . . . 106
2.6.6 TASCII85RingBuffer.FillCount . . . 106
2.6.7 TASCII85RingBuffer.Size . . . 106

3 Reference for unit 'AVL_Tree' 107
3.1 Used units . . . 107
3.2 Overview . . . 107
3.3 Constants, types and variables . . . 107
3.3.1 Types . . . 107
3.3.2 Variables . . . 108
3.4 TAVLTree . . . 108
3.4.1 Description . . . 108
3.4.2 Method overview . . . 109
3.4.3 Property overview . . . 110
3.4.4 TAVLTree.Create . . . 110
3.4.5 TAVLTree.CreateObjectCompare . . . 110
3.4.6 TAVLTree.Destroy . . . 110

3.4.7 TAVLTree.SetNodeManager . . . 110
3.4.8 TAVLTree.NewNode . . . 111
3.4.9 TAVLTree.DisposeNode . . . 111
3.4.10 TAVLTree.Add . . . 111
3.4.11 TAVLTree.AddAscendingSequence . . . 112
3.4.12 TAVLTree.Delete . . . 112
3.4.13 TAVLTree.Remove . . . 112
3.4.14 TAVLTree.RemovePointer . . . 113
3.4.15 TAVLTree.MoveDataLeftMost . . . 113
3.4.16 TAVLTree.MoveDataRightMost . . . 113
3.4.17 TAVLTree.Clear . . . 113
3.4.18 TAVLTree.FreeAndClear . . . 114
3.4.19 TAVLTree.FreeAndDelete . . . 114
3.4.20 TAVLTree.Equals . . . 114
3.4.21 TAVLTree.IsEqual . . . 114
3.4.22 TAVLTree.Assign . . . 115
3.4.23 TAVLTree.Compare . . . 115
3.4.24 TAVLTree.Find . . . 115
3.4.25 TAVLTree.FindKey . . . 115
3.4.26 TAVLTree.FindNearestKey . . . 116
3.4.27 TAVLTree.FindSuccessor . . . 116
3.4.28 TAVLTree.FindPrecessor . . . 116
3.4.29 TAVLTree.FindLowest . . . 117
3.4.30 TAVLTree.FindHighest . . . 117
3.4.31 TAVLTree.FindNearest . . . 117
3.4.32 TAVLTree.FindPointer . . . 117
3.4.33 TAVLTree.FindLeftMost . . . 118
3.4.34 TAVLTree.FindRightMost . . . 118
3.4.35 TAVLTree.FindLeftMostKey . . . 118
3.4.36 TAVLTree.FindRightMostKey . . . 118
3.4.37 TAVLTree.FindLeftMostSameKey . . . 119
3.4.38 TAVLTree.FindRightMostSameKey . . . 119
3.4.39 TAVLTree.GetEnumerator . . . 119
3.4.40 TAVLTree.GetEnumeratorHighToLow . . . 119
3.4.41 TAVLTree.ConsistencyCheck . . . 120
3.4.42 TAVLTree.WriteReportToStream . . . 120
3.4.43 TAVLTree.NodeToReportStr . . . 120
3.4.44 TAVLTree.ReportAsString . . . 120
3.4.45 TAVLTree.OnCompare . . . 121
3.4.46 TAVLTree.OnObjectCompare . . . 121

3.4.47 TAVLTree.NodeClass . . . 121
3.4.48 TAVLTree.Root . . . 121
3.4.49 TAVLTree.Count . . . 122
3.5 TAVLTreeNode . . . 122
3.5.1 Description . . . 122
3.5.2 Method overview . . . 122
3.5.3 TAVLTreeNode.Successor . . . 122
3.5.4 TAVLTreeNode.Precessor . . . 122
3.5.5 TAVLTreeNode.Clear . . . 123
3.5.6 TAVLTreeNode.TreeDepth . . . 123
3.5.7 TAVLTreeNode.ConsistencyCheck . . . 123
3.5.8 TAVLTreeNode.GetCount . . . 123
3.6 TAVLTreeNodeEnumerator . . . 123
3.6.1 Description . . . 123
3.6.2 Method overview . . . 124
3.6.3 Property overview . . . 124
3.6.4 TAVLTreeNodeEnumerator.Create . . . 124
3.6.5 TAVLTreeNodeEnumerator.GetEnumerator . . . 124
3.6.6 TAVLTreeNodeEnumerator.MoveNext . . . 124
3.6.7 TAVLTreeNodeEnumerator.Current . . . 124
3.6.8 TAVLTreeNodeEnumerator.LowToHigh . . . 125
3.7 TAVLTreeNodeMemManager . . . 125
3.7.1 Description . . . 125
3.7.2 Method overview . . . 125
3.7.3 Property overview . . . 125
3.7.4 TAVLTreeNodeMemManager.DisposeNode . . . 125
3.7.5 TAVLTreeNodeMemManager.NewNode . . . 126
3.7.6 TAVLTreeNodeMemManager.Clear . . . 126
3.7.7 TAVLTreeNodeMemManager.Create . . . 126
3.7.8 TAVLTreeNodeMemManager.Destroy . . . 126
3.7.9 TAVLTreeNodeMemManager.MinimumFreeNode . . . 127
3.7.10 TAVLTreeNodeMemManager.MaximumFreeNodeRatio . . . 127
3.7.11 TAVLTreeNodeMemManager.Count . . . 127
3.8 TBaseAVLTreeNodeManager . . . 127
3.8.1 Description . . . 127
3.8.2 Method overview . . . 128
3.8.3 TBaseAVLTreeNodeManager.DisposeNode . . . 128
3.8.4 TBaseAVLTreeNodeManager.NewNode . . . 128

4 Reference for unit 'base64' 129

4.1 Used units . . . 129
4.2 Overview . . . 129
4.3 Constants, types and variables . . . 129
4.3.1 Types . . . 129
4.4 Procedures and functions . . . 130
4.4.1 DecodeStringBase64 . . . 130
4.4.2 EncodeStringBase64 . . . 130
4.5 EBase64DecodingException . . . 130
4.5.1 Description . . . 130
4.6 TBase64DecodingStream . . . 130
4.6.1 Description . . . 130
4.6.2 Method overview . . . 131
4.6.3 Property overview . . . 131
4.6.4 TBase64DecodingStream.Create . . . 131
4.6.5 TBase64DecodingStream.Reset . . . 131
4.6.6 TBase64DecodingStream.Read . . . 131
4.6.7 TBase64DecodingStream.Seek . . . 132
4.6.8 TBase64DecodingStream.EOF . . . 132
4.6.9 TBase64DecodingStream.Mode . . . 132
4.7 TBase64EncodingStream . . . 133
4.7.1 Description . . . 133
4.7.2 Method overview . . . 133
4.7.3 TBase64EncodingStream.Destroy . . . 133
4.7.4 TBase64EncodingStream.Flush . . . 133
4.7.5 TBase64EncodingStream.Write . . . 134
4.7.6 TBase64EncodingStream.Seek . . . 134

5 Reference for unit 'BlowFish' 135
5.1 Used units . . . 135
5.2 Overview . . . 135
5.3 Constants, types and variables . . . 135
5.3.1 Constants . . . 135
5.3.2 Types . . . 135
5.4 EBlowFishError . . . 136
5.4.1 Description . . . 136
5.5 TBlowFish . . . 136
5.5.1 Description . . . 136
5.5.2 Method overview . . . 136
5.5.3 TBlowFish.Create . . . 136
5.5.4 TBlowFish.Encrypt . . . 137

5.5.5 TBlowFish.Decrypt . . . 137
5.6 TBlowFishDeCryptStream . . . 137
5.6.1 Description . . . 137
5.6.2 Method overview . . . 137
5.6.3 TBlowFishDeCryptStream.Create . . . 137
5.6.4 TBlowFishDeCryptStream.Read . . . 138
5.6.5 TBlowFishDeCryptStream.Seek . . . 138
5.7 TBlowFishEncryptStream . . . 138
5.7.1 Description . . . 138
5.7.2 Method overview . . . 139
5.7.3 TBlowFishEncryptStream.Destroy . . . 139
5.7.4 TBlowFishEncryptStream.Write . . . 139
5.7.5 TBlowFishEncryptStream.Seek . . . 139
5.7.6 TBlowFishEncryptStream.Flush . . . 140
5.8 TBlowFishStream . . . 140
5.8.1 Description . . . 140
5.8.2 Method overview . . . 140
5.8.3 Property overview . . . 140
5.8.4 TBlowFishStream.Create . . . 140
5.8.5 TBlowFishStream.Destroy . . . 141
5.8.6 TBlowFishStream.BlowFish . . . 141

6 Reference for unit 'BufDataset' 142
6.1 Used units . . . 142
6.2 Overview . . . 142
6.3 Constants, types and variables . . . 142
6.3.1 Types . . . 142
6.4 Procedures and functions . . . 145
6.4.1 RegisterDatapacketReader . . . 145
6.5 TApplyRecUpdateResult . . . 145
6.6 TBlobBuffer . . . 146
6.7 TBufBlobField . . . 146
6.8 TBufBookmark . . . 146
6.9 TBufRecLinkItem . . . 146
6.10 TDBCompareRec . . . 147
6.11 TRecUpdateBuffer . . . 147
6.12 TArrayBufIndex . . . 147
6.12.1 Description . . . 147
6.12.2 Method overview . . . 148
6.12.3 TArrayBufIndex.Create . . . 148

6.12.4 TArrayBufIndex.ScrollBackward . . . 148
6.12.5 TArrayBufIndex.ScrollForward . . . 148
6.12.6 TArrayBufIndex.GetCurrent . . . 148
6.12.7 TArrayBufIndex.ScrollFirst . . . 149
6.12.8 TArrayBufIndex.ScrollLast . . . 149
6.12.9 TArrayBufIndex.SetToFirstRecord . . . 149
6.12.10 TArrayBufIndex.SetToLastRecord . . . 149
6.12.11 TArrayBufIndex.StoreCurrentRecord . . . 149
6.12.12 TArrayBufIndex.RestoreCurrentRecord . . . 149
6.12.13 TArrayBufIndex.CanScrollForward . . . 150
6.12.14 TArrayBufIndex.DoScrollForward . . . 150
6.12.15 TArrayBufIndex.StoreCurrentRecIntoBookmark . . . 150
6.12.16 TArrayBufIndex.StoreSpareRecIntoBookmark . . . 150
6.12.17 TArrayBufIndex.GotoBookmark . . . 150
6.12.18 TArrayBufIndex.InitialiseIndex . . . 150
6.12.19 TArrayBufIndex.InitialiseSpareRecord . . . 150
6.12.20 TArrayBufIndex.ReleaseSpareRecord . . . 151
6.12.21 TArrayBufIndex.BeginUpdate . . . 151
6.12.22 TArrayBufIndex.AddRecord . . . 151
6.12.23 TArrayBufIndex.InsertRecordBeforeCurrentRecord . . . 151
6.12.24 TArrayBufIndex.RemoveRecordFromIndex . . . 151
6.12.25 TArrayBufIndex.EndUpdate . . . 151
6.13 TBufBlobStream . . . 152
6.13.1 Description . . . 152
6.13.2 Method overview . . . 152
6.13.3 TBufBlobStream.Create . . . 152
6.13.4 TBufBlobStream.Destroy . . . 152
6.14 TBufDataset . . . 152
6.14.1 Description . . . 152
6.14.2 Property overview . . . 153
6.14.3 TBufDataset.MaxIndexesCount . . . 153
6.14.4 TBufDataset.FieldDefs . . . 153
6.14.5 TBufDataset.Active . . . 154
6.14.6 TBufDataset.AutoCalcFields . . . 154
6.14.7 TBufDataset.Filter . . . 154
6.14.8 TBufDataset.Filtered . . . 154
6.14.9 TBufDataset.ReadOnly . . . 154
6.14.10 TBufDataset.AfterCancel . . . 154
6.14.11 TBufDataset.AfterClose . . . 155
6.14.12 TBufDataset.AfterDelete . . . 155

6.14.13 TBufDataset.AfterEdit . . . 155
6.14.14 TBufDataset.AfterInsert . . . 155
6.14.15 TBufDataset.AfterOpen . . . 155
6.14.16 TBufDataset.AfterPost . . . 155
6.14.17 TBufDataset.AfterScroll . . . 155
6.14.18 TBufDataset.BeforeCancel . . . 156
6.14.19 TBufDataset.BeforeClose . . . 156
6.14.20 TBufDataset.BeforeDelete . . . 156
6.14.21 TBufDataset.BeforeEdit . . . 156
6.14.22 TBufDataset.BeforeInsert . . . 156
6.14.23 TBufDataset.BeforeOpen . . . 156
6.14.24 TBufDataset.BeforePost . . . 156
6.14.25 TBufDataset.BeforeScroll . . . 157
6.14.26 TBufDataset.OnCalcFields . . . 157
6.14.27 TBufDataset.OnDeleteError . . . 157
6.14.28 TBufDataset.OnEditError . . . 157
6.14.29 TBufDataset.OnFilterRecord . . . 157
6.14.30 TBufDataset.OnNewRecord . . . 157
6.14.31 TBufDataset.OnPostError . . . 157
6.15 TBufIndex . . . 158
6.15.1 Description . . . 158
6.15.2 Method overview . . . 159
6.15.3 Property overview . . . 159
6.15.4 TBufIndex.Create . . . 160
6.15.5 TBufIndex.ScrollBackward . . . 160
6.15.6 TBufIndex.ScrollForward . . . 160
6.15.7 TBufIndex.GetCurrent . . . 160
6.15.8 TBufIndex.ScrollFirst . . . 160
6.15.9 TBufIndex.ScrollLast . . . 160
6.15.10 TBufIndex.GetRecord . . . 161
6.15.11 TBufIndex.SetToFirstRecord . . . 161
6.15.12 TBufIndex.SetToLastRecord . . . 161
6.15.13 TBufIndex.StoreCurrentRecord . . . 161
6.15.14 TBufIndex.RestoreCurrentRecord . . . 161
6.15.15 TBufIndex.CanScrollForward . . . 161
6.15.16 TBufIndex.DoScrollForward . . . 162
6.15.17 TBufIndex.StoreCurrentRecIntoBookmark . . . 162
6.15.18 TBufIndex.StoreSpareRecIntoBookmark . . . 162
6.15.19 TBufIndex.GotoBookmark . . . 162
6.15.20 TBufIndex.BookmarkValid . . . 162

6.15.21 TBufIndex.CompareBookmarks . . . 162
6.15.22 TBufIndex.SameBookmarks . . . 163
6.15.23 TBufIndex.InitialiseIndex . . . 163
6.15.24 TBufIndex.InitialiseSpareRecord . . . 163
6.15.25 TBufIndex.ReleaseSpareRecord . . . 163
6.15.26 TBufIndex.BeginUpdate . . . 163
6.15.27 TBufIndex.AddRecord . . . 164
6.15.28 TBufIndex.InsertRecordBeforeCurrentRecord . . . 164
6.15.29 TBufIndex.RemoveRecordFromIndex . . . 164
6.15.30 TBufIndex.OrderCurrentRecord . . . 164
6.15.31 TBufIndex.EndUpdate . . . 164
6.15.32 TBufIndex.SpareRecord . . . 165
6.15.33 TBufIndex.SpareBuffer . . . 165
6.15.34 TBufIndex.CurrentRecord . . . 165
6.15.35 TBufIndex.CurrentBuffer . . . 165
6.15.36 TBufIndex.IsInitialized . . . 165
6.15.37 TBufIndex.BookmarkSize . . . 166
6.15.38 TBufIndex.RecNo . . . 166
6.16 TCustomBufDataset . . . 166
6.16.1 Description . . . 166
6.16.2 Method overview . . . 167
6.16.3 Property overview . . . 168
6.16.4 TCustomBufDataset.Create . . . 168
6.16.5 TCustomBufDataset.GetFieldData . . . 169
6.16.6 TCustomBufDataset.SetFieldData . . . 169
6.16.7 TCustomBufDataset.ApplyUpdates . . . 169
6.16.8 TCustomBufDataset.MergeChangeLog . . . 170
6.16.9 TCustomBufDataset.RevertRecord . . . 170
6.16.10 TCustomBufDataset.CancelUpdates . . . 170
6.16.11 TCustomBufDataset.Destroy . . . 170
6.16.12 TCustomBufDataset.Locate . . . 171
6.16.13 TCustomBufDataset.Lookup . . . 171
6.16.14 TCustomBufDataset.UpdateStatus . . . 172
6.16.15 TCustomBufDataset.CreateBlobStream . . . 173
6.16.16 TCustomBufDataset.AddIndex . . . 173
6.16.17 TCustomBufDataset.ClearIndexes . . . 174
6.16.18 TCustomBufDataset.SetDatasetPacket . . . 174
6.16.19 TCustomBufDataset.GetDatasetPacket . . . 174
6.16.20 TCustomBufDataset.LoadFromStream . . . 175
6.16.21 TCustomBufDataset.SaveToStream . . . 175

6.16.22 TCustomBufDataset.LoadFromFile . . . 176
6.16.23 TCustomBufDataset.SaveToFile . . . 176
6.16.24 TCustomBufDataset.CreateDataset . . . 177
6.16.25 TCustomBufDataset.Clear . . . 177
6.16.26 TCustomBufDataset.BookmarkValid . . . 178
6.16.27 TCustomBufDataset.CompareBookmarks . . . 178
6.16.28 TCustomBufDataset.CopyFromDataset . . . 178
6.16.29 TCustomBufDataset.ChangeCount . . . 179
6.16.30 TCustomBufDataset.MaxIndexesCount . . . 179
6.16.31 TCustomBufDataset.ReadOnly . . . 180
6.16.32 TCustomBufDataset.ManualMergeChangeLog . . . 180
6.16.33 TCustomBufDataset.FileName . . . 180
6.16.34 TCustomBufDataset.PacketRecords . . . 181
6.16.35 TCustomBufDataset.OnUpdateError . . . 181
6.16.36 TCustomBufDataset.IndexDefs . . . 182
6.16.37 TCustomBufDataset.IndexName . . . 182
6.16.38 TCustomBufDataset.IndexFieldNames . . . 182
6.16.39 TCustomBufDataset.UniDirectional . . . 183
6.17 TDataPacketHandler . . . 183
6.17.1 Method overview . . . 183
6.17.2 TDataPacketHandler.Create . . . 184
6.17.3 TDataPacketHandler.LoadFieldDefs . . . 184
6.17.4 TDataPacketHandler.InitLoadRecords . . . 184
6.17.5 TDataPacketHandler.GetCurrentRecord . . . 184
6.17.6 TDataPacketHandler.GetRecordRowState . . . 184
6.17.7 TDataPacketHandler.RestoreRecord . . . 184
6.17.8 TDataPacketHandler.GotoNextRecord . . . 184
6.17.9 TDataPacketHandler.StoreFieldDefs . . . 184
6.17.10 TDataPacketHandler.StoreRecord . . . 185
6.17.11 TDataPacketHandler.FinalizeStoreRecords . . . 185
6.17.12 TDataPacketHandler.RecognizeStream . . . 185
6.18 TDoubleLinkedBufIndex . . . 185
6.18.1 Description . . . 185
6.18.2 Method overview . . . 186
6.18.3 TDoubleLinkedBufIndex.ScrollBackward . . . 186
6.18.4 TDoubleLinkedBufIndex.ScrollForward . . . 186
6.18.5 TDoubleLinkedBufIndex.GetCurrent . . . 186
6.18.6 TDoubleLinkedBufIndex.ScrollFirst . . . 187
6.18.7 TDoubleLinkedBufIndex.ScrollLast . . . 187
6.18.8 TDoubleLinkedBufIndex.GetRecord . . . 187

6.18.9 TDoubleLinkedBufIndex.SetToFirstRecord . . . 187
6.18.10 TDoubleLinkedBufIndex.SetToLastRecord . . . 187
6.18.11 TDoubleLinkedBufIndex.StoreCurrentRecord . . . 187
6.18.12 TDoubleLinkedBufIndex.RestoreCurrentRecord . . . 188
6.18.13 TDoubleLinkedBufIndex.CanScrollForward . . . 188
6.18.14 TDoubleLinkedBufIndex.DoScrollForward . . . 188
6.18.15 TDoubleLinkedBufIndex.StoreCurrentRecIntoBookmark . . . 188
6.18.16 TDoubleLinkedBufIndex.StoreSpareRecIntoBookmark . . . 188
6.18.17 TDoubleLinkedBufIndex.GotoBookmark . . . 188
6.18.18 TDoubleLinkedBufIndex.CompareBookmarks . . . 189
6.18.19 TDoubleLinkedBufIndex.SameBookmarks . . . 189
6.18.20 TDoubleLinkedBufIndex.InitialiseIndex . . . 189
6.18.21 TDoubleLinkedBufIndex.InitialiseSpareRecord . . . 189
6.18.22 TDoubleLinkedBufIndex.ReleaseSpareRecord . . . 189
6.18.23 TDoubleLinkedBufIndex.BeginUpdate . . . 189
6.18.24 TDoubleLinkedBufIndex.AddRecord . . . 190
6.18.25 TDoubleLinkedBufIndex.InsertRecordBeforeCurrentRecord . . . 190
6.18.26 TDoubleLinkedBufIndex.RemoveRecordFromIndex . . . 190
6.18.27 TDoubleLinkedBufIndex.OrderCurrentRecord . . . 190
6.18.28 TDoubleLinkedBufIndex.EndUpdate . . . 190
6.19 TFpcBinaryDatapacketHandler . . . 191
6.19.1 Method overview . . . 191
6.19.2 TFpcBinaryDatapacketHandler.Create . . . 191
6.19.3 TFpcBinaryDatapacketHandler.LoadFieldDefs . . . 191
6.19.4 TFpcBinaryDatapacketHandler.StoreFieldDefs . . . 191
6.19.5 TFpcBinaryDatapacketHandler.InitLoadRecords . . . 191
6.19.6 TFpcBinaryDatapacketHandler.GetCurrentRecord . . . 191
6.19.7 TFpcBinaryDatapacketHandler.GetRecordRowState . . . 191
6.19.8 TFpcBinaryDatapacketHandler.RestoreRecord . . . 192
6.19.9 TFpcBinaryDatapacketHandler.GotoNextRecord . . . 192
6.19.10 TFpcBinaryDatapacketHandler.StoreRecord . . . 192
6.19.11 TFpcBinaryDatapacketHandler.FinalizeStoreRecords . . . 192
6.19.12 TFpcBinaryDatapacketHandler.RecognizeStream . . . 192
6.20 TUniDirectionalBufIndex . . . 192
6.20.1 Description . . . 192
6.20.2 Method overview . . . 193
6.20.3 TUniDirectionalBufIndex.ScrollBackward . . . 193
6.20.4 TUniDirectionalBufIndex.ScrollForward . . . 193
6.20.5 TUniDirectionalBufIndex.GetCurrent . . . 193
6.20.6 TUniDirectionalBufIndex.ScrollFirst . . . 193

6.20.7 TUniDirectionalBufIndex.ScrollLast . . . 194
6.20.8 TUniDirectionalBufIndex.SetToFirstRecord . . . 194
6.20.9 TUniDirectionalBufIndex.SetToLastRecord . . . 194
6.20.10 TUniDirectionalBufIndex.StoreCurrentRecord . . . 194
6.20.11 TUniDirectionalBufIndex.RestoreCurrentRecord . . . 194
6.20.12 TUniDirectionalBufIndex.CanScrollForward . . . 194
6.20.13 TUniDirectionalBufIndex.DoScrollForward . . . 195
6.20.14 TUniDirectionalBufIndex.StoreCurrentRecIntoBookmark . . . 195
6.20.15 TUniDirectionalBufIndex.StoreSpareRecIntoBookmark . . . 195
6.20.16 TUniDirectionalBufIndex.GotoBookmark . . . 195
6.20.17 TUniDirectionalBufIndex.InitialiseIndex . . . 195
6.20.18 TUniDirectionalBufIndex.InitialiseSpareRecord . . . 195
6.20.19 TUniDirectionalBufIndex.ReleaseSpareRecord . . . 195
6.20.20 TUniDirectionalBufIndex.BeginUpdate . . . 196
6.20.21 TUniDirectionalBufIndex.AddRecord . . . 196
6.20.22 TUniDirectionalBufIndex.InsertRecordBeforeCurrentRecord . . . 196
6.20.23 TUniDirectionalBufIndex.RemoveRecordFromIndex . . . 196
6.20.24 TUniDirectionalBufIndex.OrderCurrentRecord . . . 196
6.20.25 TUniDirectionalBufIndex.EndUpdate . . . 196

7 Reference for unit 'bufstream' 197
7.1 Used units . . . 197
7.2 Overview . . . 197
7.3 Constants, types and variables . . . 197
7.3.1 Constants . . . 197
7.4 TBufferedFileStream . . . 198
7.4.1 Description . . . 198
7.4.2 Method overview . . . 198
7.4.3 TBufferedFileStream.Create . . . 198
7.4.4 TBufferedFileStream.Destroy . . . 199
7.4.5 TBufferedFileStream.Seek . . . 199
7.4.6 TBufferedFileStream.Read . . . 200
7.4.7 TBufferedFileStream.Write . . . 200
7.4.8 TBufferedFileStream.Flush . . . 201
7.4.9 TBufferedFileStream.InitializeCache . . . 201
7.5 TBufStream . . . 201
7.5.1 Description . . . 201
7.5.2 Method overview . . . 202
7.5.3 Property overview . . . 202
7.5.4 TBufStream.Create . . . 202

7.5.5 TBufStream.Destroy . . . 202
7.5.6 TBufStream.Buffer . . . 202
7.5.7 TBufStream.Capacity . . . 203
7.5.8 TBufStream.BufferPos . . . 203
7.5.9 TBufStream.BufferSize . . . 203
7.6 TReadBufStream . . . 203
7.6.1 Description . . . 203
7.6.2 Method overview . . . 204
7.6.3 TReadBufStream.Seek . . . 204
7.6.4 TReadBufStream.Read . . . 204
7.7 TWriteBufStream . . . 204
7.7.1 Description . . . 204
7.7.2 Method overview . . . 204
7.7.3 TWriteBufStream.Destroy . . . 205
7.7.4 TWriteBufStream.Seek . . . 205
7.7.5 TWriteBufStream.Write . . . 205

8 Reference for unit 'CacheCls' 206
8.1 Used units . . . 206
8.2 Overview . . . 206
8.3 Constants, types and variables . . . 206
8.3.1 Resource strings . . . 206
8.3.2 Types . . . 206
8.4 TCacheSlot . . . 207
8.5 ECacheError . . . 207
8.5.1 Description . . . 207
8.6 TCache . . . 207
8.6.1 Description . . . 207
8.6.2 Method overview . . . 208
8.6.3 Property overview . . . 208
8.6.4 TCache.Create . . . 208
8.6.5 TCache.Destroy . . . 208
8.6.6 TCache.Add . . . 208
8.6.7 TCache.AddNew . . . 209
8.6.8 TCache.FindSlot . . . 209
8.6.9 TCache.IndexOf . . . 209
8.6.10 TCache.Remove . . . 210
8.6.11 TCache.Data . . . 210
8.6.12 TCache.MRUSlot . . . 210
8.6.13 TCache.LRUSlot . . . 211

8.6.14 TCache.SlotCount . . . 211
8.6.15 TCache.Slots . . . 211
8.6.16 TCache.OnIsDataEqual . . . 211
8.6.17 TCache.OnFreeSlot . . . 212

9 Reference for unit 'Contnrs' 213
9.1 Used units . . . 213
9.2 Overview . . . 213
9.3 Constants, types and variables . . . 213
9.3.1 Constants . . . 213
9.3.2 Types . . . 214
9.4 Procedures and functions . . . 217
9.4.1 RSHash . . . 217
9.5 TBucket . . . 217
9.6 TBucketItem . . . 217
9.7 THashItem . . . 218
9.8 EDuplicate . . . 218
9.8.1 Description . . . 218
9.9 EKeyNotFound . . . 218
9.9.1 Description . . . 218
9.10 TBucketList . . . 218
9.10.1 Description . . . 218
9.10.2 Method overview . . . 218
9.10.3 TBucketList.Create . . . 218
9.11 TClassList . . . 219
9.11.1 Description . . . 219
9.11.2 Method overview . . . 219
9.11.3 Property overview . . . 219
9.11.4 TClassList.Add . . . 219
9.11.5 TClassList.Extract . . . 220
9.11.6 TClassList.Remove . . . 220
9.11.7 TClassList.IndexOf . . . 220
9.11.8 TClassList.First . . . 220
9.11.9 TClassList.Last . . . 221
9.11.10 TClassList.Insert . . . 221
9.11.11 TClassList.Items . . . 221
9.12 TComponentList . . . 221
9.12.1 Description . . . 221
9.12.2 Method overview . . . 222
9.12.3 Property overview . . . 222

9.12.4 TComponentList.Destroy . . . 222
9.12.5 TComponentList.Add . . . 222
9.12.6 TComponentList.Extract . . . 222
9.12.7 TComponentList.Remove . . . 223
9.12.8 TComponentList.IndexOf . . . 223
9.12.9 TComponentList.First . . . 223
9.12.10 TComponentList.Last . . . 224
9.12.11 TComponentList.Insert . . . 224
9.12.12 TComponentList.Items . . . 224
9.13 TCustomBucketList . . . 224
9.13.1 Description . . . 224
9.13.2 Method overview . . . 225
9.13.3 Property overview . . . 225
9.13.4 TCustomBucketList.Destroy . . . 225
9.13.5 TCustomBucketList.Clear . . . 225
9.13.6 TCustomBucketList.Add . . . 226
9.13.7 TCustomBucketList.Assign . . . 226
9.13.8 TCustomBucketList.Exists . . . 226
9.13.9 TCustomBucketList.Find . . . 226
9.13.10 TCustomBucketList.ForEach . . . 227
9.13.11 TCustomBucketList.Remove . . . 227
9.13.12 TCustomBucketList.Data . . . 227
9.14 TFPCustomHashTable . . . 227
9.14.1 Description . . . 227
9.14.2 Method overview . . . 228
9.14.3 Property overview . . . 228
9.14.4 TFPCustomHashTable.Create . . . 228
9.14.5 TFPCustomHashTable.CreateWith . . . 229
9.14.6 TFPCustomHashTable.Destroy . . . 229
9.14.7 TFPCustomHashTable.ChangeTableSize . . . 229
9.14.8 TFPCustomHashTable.Clear . . . 229
9.14.9 TFPCustomHashTable.Delete . . . 230
9.14.10 TFPCustomHashTable.Find . . . 230
9.14.11 TFPCustomHashTable.IsEmpty . . . 230
9.14.12 TFPCustomHashTable.HashFunction . . . 230
9.14.13 TFPCustomHashTable.Count . . . 231
9.14.14 TFPCustomHashTable.HashTableSize . . . 231
9.14.15 TFPCustomHashTable.HashTable . . . 231
9.14.16 TFPCustomHashTable.VoidSlots . . . 231
9.14.17 TFPCustomHashTable.LoadFactor . . . 232

9.14.18 TFPCustomHashTable.AVGChainLen . . . 232
9.14.19 TFPCustomHashTable.MaxChainLength . . . 232
9.14.20 TFPCustomHashTable.NumberOfCollisions . . . 232
9.14.21 TFPCustomHashTable.Density . . . 233
9.15 TFPDataHashTable . . . 233
9.15.1 Description . . . 233
9.15.2 Method overview . . . 233
9.15.3 Property overview . . . 233
9.15.4 TFPDataHashTable.Iterate . . . 233
9.15.5 TFPDataHashTable.Add . . . 234
9.15.6 TFPDataHashTable.Items . . . 234
9.16 TFPHashList . . . 234
9.16.1 Description . . . 234
9.16.2 Method overview . . . 235
9.16.3 Property overview . . . 235
9.16.4 TFPHashList.Create . . . 235
9.16.5 TFPHashList.Destroy . . . 235
9.16.6 TFPHashList.Add . . . 236
9.16.7 TFPHashList.Clear . . . 236
9.16.8 TFPHashList.NameOfIndex . . . 236
9.16.9 TFPHashList.HashOfIndex . . . 236
9.16.10 TFPHashList.GetNextCollision . . . 237
9.16.11 TFPHashList.Delete . . . 237
9.16.12 TFPHashList.Error . . . 237
9.16.13 TFPHashList.Expand . . . 237
9.16.14 TFPHashList.Extract . . . 237
9.16.15 TFPHashList.IndexOf . . . 238
9.16.16 TFPHashList.Find . . . 238
9.16.17 TFPHashList.FindIndexOf . . . 238
9.16.18 TFPHashList.FindWithHash . . . 238
9.16.19 TFPHashList.Rename . . . 239
9.16.20 TFPHashList.Remove . . . 239
9.16.21 TFPHashList.Pack . . . 239
9.16.22 TFPHashList.ShowStatistics . . . 239
9.16.23 TFPHashList.ForEachCall . . . 240
9.16.24 TFPHashList.Capacity . . . 240
9.16.25 TFPHashList.Count . . . 240
9.16.26 TFPHashList.Items . . . 240
9.16.27 TFPHashList.List . . . 241
9.16.28 TFPHashList.Strs . . . 241

9.17 TFPHashObject . . . 241
9.17.1 Description . . . 241
9.17.2 Method overview . . . 241
9.17.3 Property overview . . . 241
9.17.4 TFPHashObject.CreateNotOwned . . . 242
9.17.5 TFPHashObject.Create . . . 242
9.17.6 TFPHashObject.ChangeOwner . . . 242
9.17.7 TFPHashObject.ChangeOwnerAndName . . . 242
9.17.8 TFPHashObject.Rename . . . 243
9.17.9 TFPHashObject.Name . . . 243
9.17.10 TFPHashObject.Hash . . . 243
9.18 TFPHashObjectList . . . 244
9.18.1 Method overview . . . 244
9.18.2 Property overview . . . 244
9.18.3 TFPHashObjectList.Create . . . 244
9.18.4 TFPHashObjectList.Destroy . . . 244
9.18.5 TFPHashObjectList.Clear . . . 245
9.18.6 TFPHashObjectList.Add . . . 245
9.18.7 TFPHashObjectList.NameOfIndex . . . 245
9.18.8 TFPHashObjectList.HashOfIndex . . . 246
9.18.9 TFPHashObjectList.GetNextCollision . . . 246
9.18.10 TFPHashObjectList.Delete . . . 246
9.18.11 TFPHashObjectList.Expand . . . 246
9.18.12 TFPHashObjectList.Extract . . . 247
9.18.13 TFPHashObjectList.Remove . . . 247
9.18.14 TFPHashObjectList.IndexOf . . . 247
9.18.15 TFPHashObjectList.Find . . . 247
9.18.16 TFPHashObjectList.FindIndexOf . . . 248
9.18.17 TFPHashObjectList.FindWithHash . . . 248
9.18.18 TFPHashObjectList.Rename . . . 248
9.18.19 TFPHashObjectList.FindInstanceOf . . . 248
9.18.20 TFPHashObjectList.Pack . . . 249
9.18.21 TFPHashObjectList.ShowStatistics . . . 249
9.18.22 TFPHashObjectList.ForEachCall . . . 249
9.18.23 TFPHashObjectList.Capacity . . . 249
9.18.24 TFPHashObjectList.Count . . . 250
9.18.25 TFPHashObjectList.OwnsObjects . . . 250
9.18.26 TFPHashObjectList.Items . . . 250
9.18.27 TFPHashObjectList.List . . . 250
9.19 TFPObjectHashTable . . . 251

9.19.1 Description . . . 251
9.19.2 Method overview . . . 251
9.19.3 Property overview . . . 251
9.19.4 TFPObjectHashTable.Create . . . 251
9.19.5 TFPObjectHashTable.CreateWith . . . 251
9.19.6 TFPObjectHashTable.Iterate . . . 252
9.19.7 TFPObjectHashTable.Add . . . 252
9.19.8 TFPObjectHashTable.Items . . . 252
9.19.9 TFPObjectHashTable.OwnsObjects . . . 252
9.20 TFPObjectList . . . 253
9.20.1 Description . . . 253
9.20.2 Method overview . . . 253
9.20.3 Property overview . . . 254
9.20.4 TFPObjectList.Create . . . 254
9.20.5 TFPObjectList.Destroy . . . 254
9.20.6 TFPObjectList.Clear . . . 254
9.20.7 TFPObjectList.Add . . . 254
9.20.8 TFPObjectList.Delete . . . 255
9.20.9 TFPObjectList.Exchange . . . 255
9.20.10 TFPObjectList.Expand . . . 255
9.20.11 TFPObjectList.Extract . . . 256
9.20.12 TFPObjectList.Remove . . . 256
9.20.13 TFPObjectList.IndexOf . . . 256
9.20.14 TFPObjectList.FindInstanceOf . . . 256
9.20.15 TFPObjectList.Insert . . . 257
9.20.16 TFPObjectList.First . . . 257
9.20.17 TFPObjectList.Last . . . 257
9.20.18 TFPObjectList.Move . . . 258
9.20.19 TFPObjectList.Assign . . . 258
9.20.20 TFPObjectList.Pack . . . 258
9.20.21 TFPObjectList.Sort . . . 258
9.20.22 TFPObjectList.ForEachCall . . . 259
9.20.23 TFPObjectList.Capacity . . . 259
9.20.24 TFPObjectList.Count . . . 259
9.20.25 TFPObjectList.OwnsObjects . . . 260
9.20.26 TFPObjectList.Items . . . 260
9.20.27 TFPObjectList.List . . . 260
9.21 TFPStringHashTable . . . 260
9.21.1 Description . . . 260
9.21.2 Method overview . . . 261

9.21.3 Property overview . . . 261
9.21.4 TFPStringHashTable.Iterate . . . 261
9.21.5 TFPStringHashTable.Add . . . 261
9.21.6 TFPStringHashTable.Items . . . 261
9.22 THTCustomNode . . . 262
9.22.1 Description . . . 262
9.22.2 Method overview . . . 262
9.22.3 Property overview . . . 262
9.22.4 THTCustomNode.CreateWith . . . 262
9.22.5 THTCustomNode.HasKey . . . 262
9.22.6 THTCustomNode.Key . . . 263
9.23 THTDataNode . . . 263
9.23.1 Description . . . 263
9.23.2 Property overview . . . 263
9.23.3 THTDataNode.Data . . . 263
9.24 THTObjectNode . . . 263
9.24.1 Description . . . 263
9.24.2 Property overview . . . 264
9.24.3 THTObjectNode.Data . . . 264
9.25 THTOwnedObjectNode . . . 264
9.25.1 Description . . . 264
9.25.2 Method overview . . . 264
9.25.3 THTOwnedObjectNode.Destroy . . . 264
9.26 THTStringNode . . . 264
9.26.1 Description . . . 264
9.26.2 Property overview . . . 265
9.26.3 THTStringNode.Data . . . 265
9.27 TObjectBucketList . . . 265
9.27.1 Description . . . 265
9.27.2 Method overview . . . 265
9.27.3 Property overview . . . 265
9.27.4 TObjectBucketList.Add . . . 265
9.27.5 TObjectBucketList.Remove . . . 266
9.27.6 TObjectBucketList.Data . . . 266
9.28 TObjectList . . . 266
9.28.1 Description . . . 266
9.28.2 Method overview . . . 266
9.28.3 Property overview . . . 267
9.28.4 TObjectList.Create . . . 267
9.28.5 TObjectList.Add . . . 267

9.28.6 TObjectList.Extract . . . 267
9.28.7 TObjectList.Remove . . . 268
9.28.8 TObjectList.IndexOf . . . 268
9.28.9 TObjectList.FindInstanceOf . . . 268
9.28.10 TObjectList.Insert . . . 269
9.28.11 TObjectList.First . . . 269
9.28.12 TObjectList.Last . . . 269
9.28.13 TObjectList.OwnsObjects . . . 269
9.28.14 TObjectList.Items . . . 270
9.29 TObjectQueue . . . 270
9.29.1 Method overview . . . 270
9.29.2 TObjectQueue.Push . . . 270
9.29.3 TObjectQueue.Pop . . . 270
9.29.4 TObjectQueue.Peek . . . 271
9.30 TObjectStack . . . 271
9.30.1 Description . . . 271
9.30.2 Method overview . . . 271
9.30.3 TObjectStack.Push . . . 271
9.30.4 TObjectStack.Pop . . . 271
9.30.5 TObjectStack.Peek . . . 272
9.31 TOrderedList . . . 272
9.31.1 Description . . . 272
9.31.2 Method overview . . . 272
9.31.3 TOrderedList.Create . . . 272
9.31.4 TOrderedList.Destroy . . . 273
9.31.5 TOrderedList.Count . . . 273
9.31.6 TOrderedList.AtLeast . . . 273
9.31.7 TOrderedList.Push . . . 273
9.31.8 TOrderedList.Pop . . . 274
9.31.9 TOrderedList.Peek . . . 274
9.32 TQueue . . . 274
9.32.1 Description . . . 274
9.33 TStack . . . 274
9.33.1 Description . . . 274

10 Reference for unit 'csvdocument' 275
10.1 Used units . . . 275
10.2 Overview . . . 275
10.3 Constants, types and variables . . . 275
10.3.1 Types . . . 275

10.4 TCSVDocument . . . 276
10.4.1 Description . . . 276
10.4.2 Method overview . . . 276
10.4.3 Property overview . . . 276
10.4.4 TCSVDocument.Create . . . 276
10.4.5 TCSVDocument.Destroy . . . 277
10.4.6 TCSVDocument.LoadFromFile . . . 277
10.4.7 TCSVDocument.LoadFromStream . . . 277
10.4.8 TCSVDocument.SaveToFile . . . 278
10.4.9 TCSVDocument.SaveToStream . . . 278
10.4.10 TCSVDocument.AddRow . . . 278
10.4.11 TCSVDocument.AddCell . . . 278
10.4.12 TCSVDocument.InsertRow . . . 279
10.4.13 TCSVDocument.InsertCell . . . 279
10.4.14 TCSVDocument.RemoveRow . . . 279
10.4.15 TCSVDocument.RemoveCell . . . 279
10.4.16 TCSVDocument.HasRow . . . 280
10.4.17 TCSVDocument.HasCell . . . 280
10.4.18 TCSVDocument.IndexOfCol . . . 280
10.4.19 TCSVDocument.IndexOfRow . . . 280
10.4.20 TCSVDocument.Clear . . . 281
10.4.21 TCSVDocument.CloneRow . . . 281
10.4.22 TCSVDocument.ExchangeRows . . . 281
10.4.23 TCSVDocument.UnifyEmbeddedLineEndings . . . 281
10.4.24 TCSVDocument.RemoveTrailingEmptyCells . . . 281
10.4.25 TCSVDocument.Cells . . . 282
10.4.26 TCSVDocument.RowCount . . . 282
10.4.27 TCSVDocument.ColCount . . . 282
10.4.28 TCSVDocument.MaxColCount . . . 282
10.4.29 TCSVDocument.CSVText . . . 283

11 Reference for unit 'csvreadwrite' 284
11.1 Used units . . . 284
11.2 Overview . . . 284
11.3 Constants, types and variables . . . 284
11.3.1 Types . . . 284
11.4 Procedures and functions . . . 285
11.4.1 ChangeLineEndings . . . 285
11.5 TCSVBuilder . . . 285
11.5.1 Description . . . 285

11.5.2 Method overview . . . 285
11.5.3 Property overview . . . 286
11.5.4 TCSVBuilder.Create . . . 286
11.5.5 TCSVBuilder.Destroy . . . 286
11.5.6 TCSVBuilder.SetOutput . . . 286
11.5.7 TCSVBuilder.ResetBuilder . . . 286
11.5.8 TCSVBuilder.AppendCell . . . 287
11.5.9 TCSVBuilder.AppendRow . . . 287
11.5.10 TCSVBuilder.DefaultOutput . . . 287
11.5.11 TCSVBuilder.DefaultOutputAsString . . . 287
11.6 TCSVHandler . . . 288
11.6.1 Description . . . 288
11.6.2 Method overview . . . 288
11.6.3 Property overview . . . 288
11.6.4 TCSVHandler.Create . . . 288
11.6.5 TCSVHandler.Assign . . . 288
11.6.6 TCSVHandler.AssignCSVProperties . . . 289
11.6.7 TCSVHandler.Delimiter . . . 289
11.6.8 TCSVHandler.QuoteChar . . . 289
11.6.9 TCSVHandler.LineEnding . . . 289
11.6.10 TCSVHandler.IgnoreOuterWhitespace . . . 290
11.6.11 TCSVHandler.QuoteOuterWhitespace . . . 290
11.6.12 TCSVHandler.EqualColCountPerRow . . . 290
11.7 TCSVParser . . . 290
11.7.1 Description . . . 290
11.7.2 Method overview . . . 291
11.7.3 Property overview . . . 291
11.7.4 TCSVParser.Create . . . 291
11.7.5 TCSVParser.Destroy . . . 291
11.7.6 TCSVParser.SetSource . . . 292
11.7.7 TCSVParser.ResetParser . . . 292
11.7.8 TCSVParser.ParseNextCell . . . 292
11.7.9 TCSVParser.CurrentRow . . . 292
11.7.10 TCSVParser.CurrentCol . . . 293
11.7.11 TCSVParser.CurrentCellText . . . 293
11.7.12 TCSVParser.MaxColCount . . . 293
11.7.13 TCSVParser.FreeStream . . . 294
11.7.14 TCSVParser.BOM . . . 294
11.7.15 TCSVParser.DetectBOM . . . 294

12 Reference for unit 'CustApp' 295
12.1 Used units . . . 295
12.2 Overview . . . 295
12.3 Constants, types and variables . . . 295
12.3.1 Types . . . 295
12.3.2 Variables . . . 296
12.4 TCustomApplication . . . 296
12.4.1 Description . . . 296
12.4.2 Method overview . . . 297
12.4.3 Property overview . . . 297
12.4.4 TCustomApplication.Create . . . 297
12.4.5 TCustomApplication.Destroy . . . 298
12.4.6 TCustomApplication.HandleException . . . 298
12.4.7 TCustomApplication.Initialize . . . 298
12.4.8 TCustomApplication.Run . . . 299
12.4.9 TCustomApplication.ShowException . . . 299
12.4.10 TCustomApplication.Terminate . . . 299
12.4.11 TCustomApplication.FindOptionIndex . . . 300
12.4.12 TCustomApplication.GetOptionValue . . . 300
12.4.13 TCustomApplication.GetOptionValues . . . 301
12.4.14 TCustomApplication.HasOption . . . 301
12.4.15 TCustomApplication.CheckOptions . . . 301
12.4.16 TCustomApplication.GetNonOptions . . . 302
12.4.17 TCustomApplication.GetEnvironmentList . . . 303
12.4.18 TCustomApplication.Log . . . 303
12.4.19 TCustomApplication.ExeName . . . 303
12.4.20 TCustomApplication.HelpFile . . . 304
12.4.21 TCustomApplication.Terminated . . . 304
12.4.22 TCustomApplication.Title . . . 304
12.4.23 TCustomApplication.OnException . . . 304
12.4.24 TCustomApplication.ConsoleApplication . . . 305
12.4.25 TCustomApplication.Location . . . 305
12.4.26 TCustomApplication.Params . . . 305
12.4.27 TCustomApplication.ParamCount . . . 306
12.4.28 TCustomApplication.EnvironmentVariable . . . 306
12.4.29 TCustomApplication.OptionChar . . . 306
12.4.30 TCustomApplication.CaseSensitiveOptions . . . 306
12.4.31 TCustomApplication.StopOnException . . . 307
12.4.32 TCustomApplication.ExceptionExitCode . . . 307
12.4.33 TCustomApplication.EventLogFilter . . . 307

12.4.34 TCustomApplication.SingleInstance . . . 308
12.4.35 TCustomApplication.SingleInstanceClass . . . 308
12.4.36 TCustomApplication.SingleInstanceEnabled . . . 308

13 Reference for unit 'daemonapp' 309
13.1 Used units . . . 309
13.2 Overview . . . 309
13.3 Constants, types and variables . . . 310
13.3.1 Resource strings . . . 310
13.3.2 Types . . . 311
13.3.3 Variables . . . 315
13.4 Procedures and functions . . . 315
13.4.1 Application . . . 315
13.4.2 DaemonError . . . 315
13.4.3 RegisterDaemonApplicationClass . . . 316
13.4.4 RegisterDaemonClass . . . 316
13.4.5 RegisterDaemonMapper . . . 316
13.5 EDaemon . . . 316
13.5.1 Description . . . 316
13.6 TCustomDaemon . . . 317
13.6.1 Description . . . 317
13.6.2 Method overview . . . 317
13.6.3 Property overview . . . 317
13.6.4 TCustomDaemon.CheckControlMessages . . . 317
13.6.5 TCustomDaemon.LogMessage . . . 317
13.6.6 TCustomDaemon.ReportStatus . . . 318
13.6.7 TCustomDaemon.Definition . . . 318
13.6.8 TCustomDaemon.DaemonThread . . . 318
13.6.9 TCustomDaemon.Controller . . . 319
13.6.10 TCustomDaemon.Status . . . 319
13.6.11 TCustomDaemon.Logger . . . 319
13.7 TCustomDaemonApplication . . . 319
13.7.1 Description . . . 319
13.7.2 Method overview . . . 320
13.7.3 Property overview . . . 320
13.7.4 TCustomDaemonApplication.Create . . . 320
13.7.5 TCustomDaemonApplication.Destroy . . . 320
13.7.6 TCustomDaemonApplication.ShowException . . . 320
13.7.7 TCustomDaemonApplication.CreateDaemon . . . 321
13.7.8 TCustomDaemonApplication.StopDaemons . . . 321

13.7.9 TCustomDaemonApplication.InstallDaemons . . . 321
13.7.10 TCustomDaemonApplication.RunDaemons . . . 321
13.7.11 TCustomDaemonApplication.UnInstallDaemons . . . 322
13.7.12 TCustomDaemonApplication.ShowHelp . . . 322
13.7.13 TCustomDaemonApplication.CreateForm . . . 322
13.7.14 TCustomDaemonApplication.OnRun . . . 322
13.7.15 TCustomDaemonApplication.EventLog . . . 323
13.7.16 TCustomDaemonApplication.GUIMainLoop . . . 323
13.7.17 TCustomDaemonApplication.GuiHandle . . . 323
13.7.18 TCustomDaemonApplication.RunMode . . . 323
13.7.19 TCustomDaemonApplication.AutoRegisterMessageFile . . . 324
13.8 TCustomDaemonMapper . . . 324
13.8.1 Description . . . 324
13.8.2 Method overview . . . 324
13.8.3 Property overview . . . 324
13.8.4 TCustomDaemonMapper.Create . . . 324
13.8.5 TCustomDaemonMapper.Destroy . . . 325
13.8.6 TCustomDaemonMapper.DaemonDefs . . . 325
13.8.7 TCustomDaemonMapper.OnCreate . . . 325
13.8.8 TCustomDaemonMapper.OnDestroy . . . 326
13.8.9 TCustomDaemonMapper.OnRun . . . 326
13.8.10 TCustomDaemonMapper.OnInstall . . . 326
13.8.11 TCustomDaemonMapper.OnUnInstall . . . 326
13.9 TDaemon . . . 327
13.9.1 Description . . . 327
13.9.2 Property overview . . . 327
13.9.3 TDaemon.Definition . . . 327
13.9.4 TDaemon.Status . . . 327
13.9.5 TDaemon.OnStart . . . 327
13.9.6 TDaemon.OnStop . . . 328
13.9.7 TDaemon.OnPause . . . 328
13.9.8 TDaemon.OnContinue . . . 328
13.9.9 TDaemon.OnShutDown . . . 329
13.9.10 TDaemon.OnExecute . . . 329
13.9.11 TDaemon.BeforeInstall . . . 329
13.9.12 TDaemon.AfterInstall . . . 330
13.9.13 TDaemon.BeforeUnInstall . . . 330
13.9.14 TDaemon.AfterUnInstall . . . 330
13.9.15 TDaemon.OnControlCode . . . 330
13.9.16 TDaemon.OnControlCodeEvent . . . 331

13.10TDaemonApplication . . . 331
13.10.1 Description . . . 331
13.11TDaemonController . . . 331
13.11.1 Description . . . 331
13.11.2 Method overview . . . 331
13.11.3 Property overview . . . 331
13.11.4 TDaemonController.Create . . . 332
13.11.5 TDaemonController.Destroy . . . 332
13.11.6 TDaemonController.StartService . . . 332
13.11.7 TDaemonController.Main . . . 332
13.11.8 TDaemonController.Controller . . . 333
13.11.9 TDaemonController.ReportStatus . . . 333
13.11.10TDaemonController.Daemon . . . 333
13.11.11TDaemonController.Params . . . 333
13.11.12TDaemonController.LastStatus . . . 334
13.11.13TDaemonController.CheckPoint . . . 334
13.12TDaemonDef . . . 334
13.12.1 Description . . . 334
13.12.2 Method overview . . . 334
13.12.3 Property overview . . . 335
13.12.4 TDaemonDef.Create . . . 335
13.12.5 TDaemonDef.Destroy . . . 335
13.12.6 TDaemonDef.DaemonClass . . . 335
13.12.7 TDaemonDef.Instance . . . 336
13.12.8 TDaemonDef.DaemonClassName . . . 336
13.12.9 TDaemonDef.Name . . . 336
13.12.10TDaemonDef.Description . . . 336
13.12.11TDaemonDef.DisplayName . . . 337
13.12.12TDaemonDef.RunArguments . . . 337
13.12.13TDaemonDef.Options . . . 337
13.12.14TDaemonDef.Enabled . . . 337
13.12.15TDaemonDef.WinBindings . . . 338
13.12.16TDaemonDef.OnCreateInstance . . . 338
13.12.17TDaemonDef.LogStatusReport . . . 338
13.13TDaemonDefs . . . 338
13.13.1 Description . . . 338
13.13.2 Method overview . . . 339
13.13.3 Property overview . . . 339
13.13.4 TDaemonDefs.Create . . . 339
13.13.5 TDaemonDefs.IndexOfDaemonDef . . . 339

13.13.6 TDaemonDefs.FindDaemonDef . . . 339
13.13.7 TDaemonDefs.DaemonDefByName . . . 340
13.13.8 TDaemonDefs.Daemons . . . 340
13.14TDaemonMapper . . . 340
13.14.1 Description . . . 340
13.14.2 Method overview . . . 340
13.14.3 TDaemonMapper.Create . . . 341
13.14.4 TDaemonMapper.CreateNew . . . 341
13.15TDaemonThread . . . 341
13.15.1 Description . . . 341
13.15.2 Method overview . . . 341
13.15.3 Property overview . . . 341
13.15.4 TDaemonThread.Create . . . 342
13.15.5 TDaemonThread.Execute . . . 342
13.15.6 TDaemonThread.CheckControlMessage . . . 342
13.15.7 TDaemonThread.StopDaemon . . . 342
13.15.8 TDaemonThread.PauseDaemon . . . 343
13.15.9 TDaemonThread.ContinueDaemon . . . 343
13.15.10TDaemonThread.ShutDownDaemon . . . 343
13.15.11TDaemonThread.InterrogateDaemon . . . 343
13.15.12TDaemonThread.Daemon . . . 344
13.16TDependencies . . . 344
13.16.1 Description . . . 344
13.16.2 Method overview . . . 344
13.16.3 Property overview . . . 344
13.16.4 TDependencies.Create . . . 344
13.16.5 TDependencies.Items . . . 344
13.17TDependency . . . 345
13.17.1 Description . . . 345
13.17.2 Method overview . . . 345
13.17.3 Property overview . . . 345
13.17.4 TDependency.Assign . . . 345
13.17.5 TDependency.Name . . . 345
13.17.6 TDependency.IsGroup . . . 345
13.18TWinBindings . . . 346
13.18.1 Description . . . 346
13.18.2 Method overview . . . 346
13.18.3 Property overview . . . 346
13.18.4 TWinBindings.Create . . . 346
13.18.5 TWinBindings.Destroy . . . 346

13.18.6 TWinBindings.Assign . . . 347
13.18.7 TWinBindings.ErrCode . . . 347
13.18.8 TWinBindings.Win32ErrCode . . . 347
13.18.9 TWinBindings.Dependencies . . . 347
13.18.10 TWinBindings.GroupName . . . 348
13.18.11 TWinBindings.Password . . . 348
13.18.12 TWinBindings.UserName . . . 348
13.18.13 TWinBindings.StartType . . . 348
13.18.14 TWinBindings.WaitHint . . . 349
13.18.15 TWinBindings.IDTag . . . 349
13.18.16 TWinBindings.ServiceType . . . 349
13.18.17 TWinBindings.ErrorSeverity . . . 349
13.18.18 TWinBindings.AcceptedCodes . . . 350

14 Reference for unit 'DB' 351
14.1 Used units . . . 351
14.2 Overview . . . 351
14.3 Constants, types and variables . . . 351
14.3.1 Constants . . . 351
14.3.2 Types . . . 353
14.3.3 Variables . . . 368
14.4 Procedures and functions . . . 369
14.4.1 BuffersEqual . . . 369
14.4.2 DatabaseError . . . 369
14.4.3 DatabaseErrorFmt . . . 369
14.4.4 DateTimeRecToDateTime . . . 370
14.4.5 DateTimeToDateTimeRec . . . 370
14.4.6 DisposeMem . . . 370
14.4.7 enumerator(TDataSet):TDataSetEnumerator . . . 370
14.4.8 ExtractFieldName . . . 371
14.4.9 SkipComments . . . 371
14.5 TLookupListRec . . . 371
14.6 EDatabaseError . . . 371
14.6.1 Description . . . 371
14.7 EUpdateError . . . 372
14.7.1 Description . . . 372
14.7.2 Method overview . . . 372
14.7.3 Property overview . . . 372
14.7.4 EUpdateError.Create . . . 372
14.7.5 EUpdateError.Destroy . . . 372

14.7.6 EUpdateError.Context . . . 373
14.7.7 EUpdateError.ErrorCode . . . 373
14.7.8 EUpdateError.OriginalException . . . 373
14.7.9 EUpdateError.PreviousError . . . 373
14.8 IProviderSupport . . . 374
14.8.1 Description . . . 374
14.8.2 Method overview . . . 374
14.8.3 IProviderSupport.PSEndTransaction . . . 374
14.8.4 IProviderSupport.PSExecute . . . 374
14.8.5 IProviderSupport.PSExecuteStatement . . . 375
14.8.6 IProviderSupport.PSGetAttributes . . . 375
14.8.7 IProviderSupport.PSGetCommandText . . . 375
14.8.8 IProviderSupport.PSGetCommandType . . . 376
14.8.9 IProviderSupport.PSGetDefaultOrder . . . 376
14.8.10 IProviderSupport.PSGetIndexDefs . . . 376
14.8.11 IProviderSupport.PSGetKeyFields . . . 376
14.8.12 IProviderSupport.PSGetParams . . . 377
14.8.13 IProviderSupport.PSGetQuoteChar . . . 377
14.8.14 IProviderSupport.PSGetTableName . . . 377
14.8.15 IProviderSupport.PSGetUpdateException . . . 377
14.8.16 IProviderSupport.PSInTransaction . . . 378
14.8.17 IProviderSupport.PSIsSQLBased . . . 378
14.8.18 IProviderSupport.PSIsSQLSupported . . . 378
14.8.19 IProviderSupport.PSReset . . . 378
14.8.20 IProviderSupport.PSSetCommandText . . . 379
14.8.21 IProviderSupport.PSSetParams . . . 379
14.8.22 IProviderSupport.PSStartTransaction . . . 379
14.8.23 IProviderSupport.PSUpdateRecord . . . 379
14.9 TArrayField . . . 380
14.9.1 Method overview . . . 380
14.9.2 TArrayField.Create . . . 380
14.10 TAutoIncField . . . 380
14.10.1 Description . . . 380
14.10.2 Method overview . . . 380
14.10.3 TAutoIncField.Create . . . 380
14.11 TBCDField . . . 380
14.11.1 Description . . . 380
14.11.2 Method overview . . . 381
14.11.3 Property overview . . . 381
14.11.4 TBCDField.Create . . . 381

14.11.5 TBCDField.CheckRange . . . 381
14.11.6 TBCDField.Value . . . 382
14.11.7 TBCDField.Precision . . . 382
14.11.8 TBCDField.Currency . . . 382
14.11.9 TBCDField.MaxValue . . . 382
14.11.10TBCDField.MinValue . . . 383
14.11.11TBCDField.Size . . . 383
14.12TBinaryField . . . 383
14.12.1 Description . . . 383
14.12.2 Method overview . . . 384
14.12.3 Property overview . . . 384
14.12.4 TBinaryField.Create . . . 384
14.12.5 TBinaryField.Size . . . 384
14.13TBlobField . . . 384
14.13.1 Description . . . 384
14.13.2 Method overview . . . 385
14.13.3 Property overview . . . 385
14.13.4 TBlobField.Create . . . 385
14.13.5 TBlobField.Clear . . . 385
14.13.6 TBlobField.IsBlob . . . 385
14.13.7 TBlobField.LoadFromFile . . . 386
14.13.8 TBlobField.LoadFromStream . . . 386
14.13.9 TBlobField.SaveToFile . . . 386
14.13.10TBlobField.SaveToStream . . . 387
14.13.11TBlobField.SetFieldType . . . 387
14.13.12TBlobField.BlobSize . . . 387
14.13.13TBlobField.Modified . . . 387
14.13.14TBlobField.Value . . . 388
14.13.15TBlobField.Transliterate . . . 388
14.13.16TBlobField.BlobType . . . 388
14.13.17TBlobField.Size . . . 388
14.14TBooleanField . . . 389
14.14.1 Description . . . 389
14.14.2 Method overview . . . 389
14.14.3 Property overview . . . 389
14.14.4 TBooleanField.Create . . . 389
14.14.5 TBooleanField.Value . . . 389
14.14.6 TBooleanField.DisplayValues . . . 390
14.15TByteField . . . 390
14.15.1 Description . . . 390

14.15.2 Method overview . . . 390
14.15.3 TByteField.Create . . . 390
14.16TBytesField . . . 391
14.16.1 Description . . . 391
14.16.2 Method overview . . . 391
14.16.3 TBytesField.Create . . . 391
14.17TCheckConstraint . . . 391
14.17.1 Description . . . 391
14.17.2 Method overview . . . 391
14.17.3 Property overview . . . 392
14.17.4 TCheckConstraint.Assign . . . 392
14.17.5 TCheckConstraint.CustomConstraint . . . 392
14.17.6 TCheckConstraint.ErrorMessage . . . 392
14.17.7 TCheckConstraint.FromDictionary . . . 393
14.17.8 TCheckConstraint.ImportedConstraint . . . 393
14.18TCheckConstraints . . . 393
14.18.1 Description . . . 393
14.18.2 Method overview . . . 393
14.18.3 Property overview . . . 393
14.18.4 TCheckConstraints.Create . . . 394
14.18.5 TCheckConstraints.Add . . . 394
14.18.6 TCheckConstraints.Items . . . 394
14.19TCurrencyField . . . 394
14.19.1 Description . . . 394
14.19.2 Method overview . . . 395
14.19.3 Property overview . . . 395
14.19.4 TCurrencyField.Create . . . 395
14.19.5 TCurrencyField.Currency . . . 395
14.20TCustomConnection . . . 395
14.20.1 Description . . . 395
14.20.2 Method overview . . . 395
14.20.3 Property overview . . . 396
14.20.4 TCustomConnection.Close . . . 396
14.20.5 TCustomConnection.Destroy . . . 396
14.20.6 TCustomConnection.Open . . . 396
14.20.7 TCustomConnection.DataSetCount . . . 397
14.20.8 TCustomConnection.DataSets . . . 397
14.20.9 TCustomConnection.Connected . . . 397
14.20.10TCustomConnection.LoginPrompt . . . 398
14.20.11TCustomConnection.AfterConnect . . . 398

14.20.12TCustomConnection.AfterDisconnect . . . 398
14.20.13TCustomConnection.BeforeConnect . . . 399
14.20.14TCustomConnection.BeforeDisconnect . . . 399
14.20.15TCustomConnection.OnLogin . . . 399
14.20.16TCustomConnection.OnCloseError . . . 399
14.21TDatabase . . . 400
14.21.1 Description . . . 400
14.21.2 Method overview . . . 400
14.21.3 Property overview . . . 400
14.21.4 TDatabase.Create . . . 400
14.21.5 TDatabase.Destroy . . . 401
14.21.6 TDatabase.CloseDataSets . . . 401
14.21.7 TDatabase.CloseTransactions . . . 401
14.21.8 TDatabase.StartTransaction . . . 401
14.21.9 TDatabase.EndTransaction . . . 402
14.21.10TDatabase.TransactionCount . . . 402
14.21.11TDatabase.Transactions . . . 402
14.21.12TDatabase.Directory . . . 402
14.21.13TDatabase.IsSQLBased . . . 403
14.21.14TDatabase.Connected . . . 403
14.21.15TDatabase.DatabaseName . . . 403
14.21.16TDatabase.KeepConnection . . . 403
14.21.17TDatabase.Params . . . 404
14.22TDataLink . . . 404
14.22.1 Description . . . 404
14.22.2 Method overview . . . 404
14.22.3 Property overview . . . 405
14.22.4 TDataLink.Create . . . 405
14.22.5 TDataLink.Destroy . . . 405
14.22.6 TDataLink.Edit . . . 405
14.22.7 TDataLink.UpdateRecord . . . 406
14.22.8 TDataLink.ExecuteAction . . . 406
14.22.9 TDataLink.UpdateAction . . . 406
14.22.10TDataLink.Active . . . 406
14.22.11TDataLink.ActiveRecord . . . 407
14.22.12TDataLink.BOF . . . 407
14.22.13TDataLink.BufferCount . . . 407
14.22.14TDataLink.DataSet . . . 408
14.22.15TDataLink.DataSource . . . 408
14.22.16TDataLink.DataSourceFixed . . . 408

14.22.17TDataLink.Editing . . . 408
14.22.18TDataLink.Eof . . . 409
14.22.19TDataLink.ReadOnly . . . 409
14.22.20TDataLink.RecordCount . . . 409
14.23TDataSet . . . 409
14.23.1 Description . . . 409
14.23.2 Method overview . . . 412
14.23.3 Property overview . . . 414
14.23.4 TDataSet.Create . . . 415
14.23.5 TDataSet.Destroy . . . 415
14.23.6 TDataSet.ActiveBuffer . . . 415
14.23.7 TDataSet.GetFieldData . . . 415
14.23.8 TDataSet.SetFieldData . . . 416
14.23.9 TDataSet.Append . . . 416
14.23.10TDataSet.AppendRecord . . . 416
14.23.11TDataSet.BookmarkValid . . . 417
14.23.12TDataSet.Cancel . . . 417
14.23.13TDataSet.CheckBrowseMode . . . 417
14.23.14TDataSet.ClearFields . . . 417
14.23.15TDataSet.Close . . . 418
14.23.16TDataSet.ControlsDisabled . . . 418
14.23.17TDataSet.CompareBookmarks . . . 418
14.23.18TDataSet.CreateBlobStream . . . 419
14.23.19TDataSet.CursorPosChanged . . . 419
14.23.20TDataSet.DataConvert . . . 419
14.23.21TDataSet.Delete . . . 419
14.23.22TDataSet.DisableControls . . . 420
14.23.23TDataSet.Edit . . . 420
14.23.24TDataSet.EnableControls . . . 421
14.23.25TDataSet.FieldByName . . . 421
14.23.26TDataSet.FindField . . . 421
14.23.27TDataSet.FindFirst . . . 422
14.23.28TDataSet.FindLast . . . 422
14.23.29TDataSet.FindNext . . . 422
14.23.30TDataSet.FindPrior . . . 422
14.23.31TDataSet.First . . . 423
14.23.32TDataSet.FreeBookmark . . . 423
14.23.33TDataSet.GetBookmark . . . 423
14.23.34TDataSet.GetCurrentRecord . . . 424
14.23.35TDataSet.GetFieldList . . . 424

14.23.36 TDataSet.GetFieldNames . . . 424
14.23.37 TDataSet.GotoBookmark . . . 424
14.23.38 TDataSet.Insert . . . 425
14.23.39 TDataSet.InsertRecord . . . 425
14.23.40 TDataSet.IsEmpty . . . 425
14.23.41 TDataSet.IsLinkedTo . . . 425
14.23.42 TDataSet.IsSequenced . . . 426
14.23.43 TDataSet.Last . . . 426
14.23.44 TDataSet.Locate . . . 426
14.23.45 TDataSet.Lookup . . . 427
14.23.46 TDataSet.MoveBy . . . 427
14.23.47 TDataSet.Next . . . 427
14.23.48 TDataSet.Open . . . 428
14.23.49 TDataSet.Post . . . 428
14.23.50 TDataSet.Prior . . . 429
14.23.51 TDataSet.Refresh . . . 429
14.23.52 TDataSet.Resync . . . 429
14.23.53 TDataSet.SetFields . . . 429
14.23.54 TDataSet.Translate . . . 430
14.23.55 TDataSet.UpdateCursorPos . . . 430
14.23.56 TDataSet.UpdateRecord . . . 430
14.23.57 TDataSet.UpdateStatus . . . 431
14.23.58 TDataSet.BlockReadSize . . . 431
14.23.59 TDataSet.BOF . . . 431
14.23.60 TDataSet.Bookmark . . . 431
14.23.61 TDataSet.CanModify . . . 432
14.23.62 TDataSet.DataSource . . . 433
14.23.63 TDataSet.DefaultFields . . . 433
14.23.64 TDataSet.EOF . . . 433
14.23.65 TDataSet.FieldCount . . . 434
14.23.66 TDataSet.FieldDefs . . . 434
14.23.67 TDataSet.Found . . . 435
14.23.68 TDataSet.Modified . . . 435
14.23.69 TDataSet.IsUniDirectional . . . 435
14.23.70 TDataSet.RecordCount . . . 436
14.23.71 TDataSet.RecNo . . . 436
14.23.72 TDataSet.RecordSize . . . 436
14.23.73 TDataSet.SparseArrays . . . 437
14.23.74 TDataSet.State . . . 437
14.23.75 TDataSet.Fields . . . 437

14.23.76TDataSet.FieldValues . . . 438
14.23.77TDataSet.Filter . . . 438
14.23.78TDataSet.Filtered . . . 438
14.23.79TDataSet.FilterOptions . . . 439
14.23.80TDataSet.Active . . . 439
14.23.81TDataSet.AutoCalcFields . . . 439
14.23.82TDataSet.BeforeOpen . . . 440
14.23.83TDataSet.AfterOpen . . . 440
14.23.84TDataSet.BeforeClose . . . 440
14.23.85TDataSet.AfterClose . . . 440
14.23.86TDataSet.BeforeInsert . . . 441
14.23.87TDataSet.AfterInsert . . . 441
14.23.88TDataSet.BeforeEdit . . . 441
14.23.89TDataSet.AfterEdit . . . 442
14.23.90TDataSet.BeforePost . . . 442
14.23.91TDataSet.AfterPost . . . 442
14.23.92TDataSet.BeforeCancel . . . 443
14.23.93TDataSet.AfterCancel . . . 443
14.23.94TDataSet.BeforeDelete . . . 443
14.23.95TDataSet.AfterDelete . . . 444
14.23.96TDataSet.BeforeScroll . . . 444
14.23.97TDataSet.AfterScroll . . . 444
14.23.98TDataSet.BeforeRefresh . . . 445
14.23.99TDataSet.AfterRefresh . . . 445
14.23.100DataSet.OnCalcFields . . . 445
14.23.101DataSet.OnDeleteError . . . 446
14.23.102DataSet.OnEditError . . . 446
14.23.103DataSet.OnFilterRecord . . . 447
14.23.104DataSet.OnNewRecord . . . 447
14.23.105DataSet.OnPostError . . . 447
14.24TDataSetEnumerator . . . 448
14.24.1 Description . . . 448
14.24.2 Method overview . . . 448
14.24.3 Property overview . . . 448
14.24.4 TDataSetEnumerator.Create . . . 448
14.24.5 TDataSetEnumerator.MoveNext . . . 448
14.24.6 TDataSetEnumerator.Current . . . 449
14.25TDataSource . . . 449
14.25.1 Description . . . 449
14.25.2 Method overview . . . 449

14.25.3 Property overview . . . 449
14.25.4 TDataSource.Create . . . 450
14.25.5 TDataSource.Destroy . . . 450
14.25.6 TDataSource.Edit . . . 450
14.25.7 TDataSource.IsLinkedTo . . . 450
14.25.8 TDataSource.State . . . 451
14.25.9 TDataSource.AutoEdit . . . 451
14.25.10TDataSource.DataSet . . . 451
14.25.11TDataSource.Enabled . . . 452
14.25.12TDataSource.OnStateChange . . . 452
14.25.13TDataSource.OnDataChange . . . 452
14.25.14TDataSource.OnUpdateData . . . 453
14.26TDateField . . . 453
14.26.1 Description . . . 453
14.26.2 Method overview . . . 453
14.26.3 TDateField.Create . . . 453
14.27TDateTimeField . . . 453
14.27.1 Description . . . 453
14.27.2 Method overview . . . 454
14.27.3 Property overview . . . 454
14.27.4 TDateTimeField.Create . . . 454
14.27.5 TDateTimeField.Value . . . 454
14.27.6 TDateTimeField.DisplayFormat . . . 454
14.27.7 TDateTimeField.EditMask . . . 455
14.28TDBDataset . . . 455
14.28.1 Description . . . 455
14.28.2 Method overview . . . 455
14.28.3 Property overview . . . 455
14.28.4 TDBDataset.destroy . . . 456
14.28.5 TDBDataset.DataBase . . . 456
14.28.6 TDBDataset.Transaction . . . 456
14.29TDBTransaction . . . 456
14.29.1 Description . . . 456
14.29.2 Method overview . . . 457
14.29.3 Property overview . . . 457
14.29.4 TDBTransaction.Create . . . 457
14.29.5 TDBTransaction.Destroy . . . 457
14.29.6 TDBTransaction.CloseDataSets . . . 457
14.29.7 TDBTransaction.DataBase . . . 458
14.29.8 TDBTransaction.Active . . . 458

14.30TDefCollection . . . 458
14.30.1 Description . . . 458
14.30.2 Method overview . . . 458
14.30.3 Property overview . . . 458
14.30.4 TDefCollection.create . . . 459
14.30.5 TDefCollection.Find . . . 459
14.30.6 TDefCollection.GetItemNames . . . 459
14.30.7 TDefCollection.IndexOf . . . 459
14.30.8 TDefCollection.Dataset . . . 460
14.30.9 TDefCollection.Updated . . . 460
14.31TDetailDataLink . . . 460
14.31.1 Description . . . 460
14.31.2 Property overview . . . 460
14.31.3 TDetailDataLink.DetailDataSet . . . 460
14.32TExtendedField . . . 461
14.32.1 Method overview . . . 461
14.32.2 Property overview . . . 461
14.32.3 TExtendedField.Create . . . 461
14.32.4 TExtendedField.CheckRange . . . 461
14.32.5 TExtendedField.Value . . . 461
14.32.6 TExtendedField.Currency . . . 461
14.32.7 TExtendedField.MaxValue . . . 461
14.32.8 TExtendedField.MinValue . . . 462
14.32.9 TExtendedField.Precision . . . 462
14.33TField . . . 462
14.33.1 Description . . . 462
14.33.2 Method overview . . . 462
14.33.3 Property overview . . . 464
14.33.4 TField.Create . . . 465
14.33.5 TField.Destroy . . . 465
14.33.6 TField.Assign . . . 465
14.33.7 TField.AssignValue . . . 465
14.33.8 TField.Clear . . . 466
14.33.9 TField.FocusControl . . . 466
14.33.10TField.GetData . . . 466
14.33.11TField.IsBlob . . . 467
14.33.12TField.IsValidChar . . . 467
14.33.13TField.RefreshLookupList . . . 467
14.33.14TField.SetData . . . 467
14.33.15TField.SetFieldType . . . 468

14.33.16 TField.Validate . . . 468
14.33.17 TField.AsBCD . . . 468
14.33.18 TField.AsBoolean . . . 469
14.33.19 TField.AsBytes . . . 469
14.33.20 TField.AsCurrency . . . 469
14.33.21 TField.AsDateTime . . . 470
14.33.22 TField.AsExtended . . . 470
14.33.23 TField.AsFloat . . . 470
14.33.24 TField.AsLongint . . . 470
14.33.25 TField.AsLongWord . . . 471
14.33.26 TField.AsLargeInt . . . 471
14.33.27 TField.AsInteger . . . 471
14.33.28 TField.AsSingle . . . 472
14.33.29 TField.AsString . . . 472
14.33.30 TField.AsAnsiString . . . 472
14.33.31 TField.AsUnicodeString . . . 473
14.33.32 TField.AsUTF8String . . . 473
14.33.33 TField.AsWideString . . . 473
14.33.34 TField.AsVariant . . . 474
14.33.35 TField.AttributeSet . . . 474
14.33.36 TField.Calculated . . . 474
14.33.37 TField.CanModify . . . 474
14.33.38 TField.CurValue . . . 475
14.33.39 TField.DataSet . . . 475
14.33.40 TField.DataSize . . . 475
14.33.41 TField.DataType . . . 475
14.33.42 TField.DisplayName . . . 476
14.33.43 TField.DisplayText . . . 476
14.33.44 TField.EditMask . . . 476
14.33.45 TField.EditMaskPtr . . . 477
14.33.46 TField.FieldNo . . . 477
14.33.47 TField.IsIndexField . . . 477
14.33.48 TField.IsNull . . . 477
14.33.49 TField.NewValue . . . 478
14.33.50 TField.Offset . . . 478
14.33.51 TField.Size . . . 478
14.33.52 TField.Text . . . 478
14.33.53 TField.ValidChars . . . 479
14.33.54 TField.Value . . . 479
14.33.55 TField.OldValue . . . 479

14.33.56TField.LookupList . . . 480
14.33.57TField.FieldDef . . . 480
14.33.58TField.Alignment . . . 480
14.33.59TField.CustomConstraint . . . 481
14.33.60TField.ConstraintErrorMessage . . . 481
14.33.61TField.DefaultExpression . . . 481
14.33.62TField.DisplayLabel . . . 482
14.33.63TField.DisplayWidth . . . 482
14.33.64TField.FieldKind . . . 482
14.33.65TField.FieldName . . . 482
14.33.66TField.HasConstraints . . . 483
14.33.67TField.Index . . . 483
14.33.68TField.ImportedConstraint . . . 483
14.33.69TField.KeyFields . . . 484
14.33.70TField.LookupCache . . . 484
14.33.71TField.LookupDataSet . . . 484
14.33.72TField.LookupKeyFields . . . 484
14.33.73TField.LookupResultField . . . 485
14.33.74TField.Lookup . . . 485
14.33.75TField.Origin . . . 485
14.33.76TField.ParentField . . . 485
14.33.77TField.ProviderFlags . . . 486
14.33.78TField.ReadOnly . . . 486
14.33.79TField.Required . . . 486
14.33.80TField.Visible . . . 487
14.33.81TField.OnChange . . . 487
14.33.82TField.OnGetText . . . 487
14.33.83TField.OnSetText . . . 487
14.33.84TField.OnValidate . . . 488
14.34TFieldDef . . . 488
14.34.1 Description . . . 488
14.34.2 Method overview . . . 488
14.34.3 Property overview . . . 489
14.34.4 TFieldDef.Create . . . 489
14.34.5 TFieldDef.Destroy . . . 489
14.34.6 TFieldDef.AddChild . . . 490
14.34.7 TFieldDef.Assign . . . 490
14.34.8 TFieldDef.CreateField . . . 490
14.34.9 TFieldDef.HasChildDefs . . . 490
14.34.10TFieldDef.FieldClass . . . 490

14.34.11TFieldDef.FieldNo . . . 491
14.34.12TFieldDef.CharSize . . . 491
14.34.13TFieldDef.InternalCalcField . . . 491
14.34.14TFieldDef.ParentDef . . . 492
14.34.15TFieldDef.Required . . . 492
14.34.16TFieldDef.Codepage . . . 492
14.34.17TFieldDef.Attributes . . . 492
14.34.18TFieldDef.DataType . . . 493
14.34.19TFieldDef.ChildDefs . . . 493
14.34.20TFieldDef.Precision . . . 493
14.34.21TFieldDef.Size . . . 493
14.35TFieldDefs . . . 494
14.35.1 Description . . . 494
14.35.2 Method overview . . . 494
14.35.3 Property overview . . . 494
14.35.4 TFieldDefs.Create . . . 494
14.35.5 TFieldDefs.Add . . . 494
14.35.6 TFieldDefs.AddFieldDef . . . 495
14.35.7 TFieldDefs.Assign . . . 495
14.35.8 TFieldDefs.Find . . . 495
14.35.9 TFieldDefs.Update . . . 496
14.35.10TFieldDefs.MakeNameUnique . . . 496
14.35.11TFieldDefs.HiddenFields . . . 496
14.35.12TFieldDefs.Items . . . 496
14.35.13TFieldDefs.ParentDef . . . 497
14.36TFields . . . 497
14.36.1 Description . . . 497
14.36.2 Method overview . . . 497
14.36.3 Property overview . . . 497
14.36.4 TFields.Create . . . 497
14.36.5 TFields.Destroy . . . 498
14.36.6 TFields.Add . . . 498
14.36.7 TFields.CheckFieldName . . . 498
14.36.8 TFields.CheckFieldNames . . . 498
14.36.9 TFields.Clear . . . 499
14.36.10TFields.FindField . . . 499
14.36.11TFields.FieldByName . . . 499
14.36.12TFields.FieldByNumber . . . 499
14.36.13TFields.GetEnumerator . . . 500
14.36.14TFields.GetFieldNames . . . 500

14.36.15TFields.IndexOf . . . 500
14.36.16TFields.Remove . . . 500
14.36.17TFields.Count . . . 501
14.36.18TFields.Dataset . . . 501
14.36.19TFields.Fields . . . 501
14.37TFieldsEnumerator . . . 501
14.37.1 Description . . . 501
14.37.2 Method overview . . . 502
14.37.3 Property overview . . . 502
14.37.4 TFieldsEnumerator.Create . . . 502
14.37.5 TFieldsEnumerator.MoveNext . . . 502
14.37.6 TFieldsEnumerator.Current . . . 503
14.38TFloatField . . . 503
14.38.1 Description . . . 503
14.38.2 Method overview . . . 503
14.38.3 Property overview . . . 503
14.38.4 TFloatField.Create . . . 503
14.38.5 TFloatField.CheckRange . . . 504
14.38.6 TFloatField.Value . . . 504
14.38.7 TFloatField.Currency . . . 504
14.38.8 TFloatField.MaxValue . . . 504
14.38.9 TFloatField.MinValue . . . 505
14.38.10TFloatField.Precision . . . 505
14.39TFMTBCDField . . . 505
14.39.1 Description . . . 505
14.39.2 Method overview . . . 506
14.39.3 Property overview . . . 506
14.39.4 TFMTBCDField.Create . . . 506
14.39.5 TFMTBCDField.CheckRange . . . 506
14.39.6 TFMTBCDField.Value . . . 506
14.39.7 TFMTBCDField.Precision . . . 507
14.39.8 TFMTBCDField.Currency . . . 507
14.39.9 TFMTBCDField.MaxValue . . . 507
14.39.10TFMTBCDField.MinValue . . . 507
14.39.11TFMTBCDField.Size . . . 508
14.40TGraphicField . . . 508
14.40.1 Description . . . 508
14.40.2 Method overview . . . 508
14.40.3 TGraphicField.Create . . . 508
14.41TGuidField . . . 508

14.41.1 Description . . . 508
14.41.2 Method overview . . . 509
14.41.3 Property overview . . . 509
14.41.4 TGuidField.Create . . . 509
14.41.5 TGuidField.AsGuid . . . 509
14.42 TIndexDef . . . 509
14.42.1 Description . . . 509
14.42.2 Method overview . . . 510
14.42.3 Property overview . . . 510
14.42.4 TIndexDef.Create . . . 510
14.42.5 TIndexDef.Expression . . . 510
14.42.6 TIndexDef.Fields . . . 510
14.42.7 TIndexDef.CaseInsFields . . . 511
14.42.8 TIndexDef.DescFields . . . 511
14.42.9 TIndexDef.Options . . . 511
14.42.10 TIndexDef.Source . . . 512
14.43 TIndexDefs . . . 512
14.43.1 Description . . . 512
14.43.2 Method overview . . . 512
14.43.3 Property overview . . . 512
14.43.4 TIndexDefs.Create . . . 512
14.43.5 TIndexDefs.Add . . . 513
14.43.6 TIndexDefs.AddIndexDef . . . 513
14.43.7 TIndexDefs.Find . . . 513
14.43.8 TIndexDefs.FindIndexForFields . . . 513
14.43.9 TIndexDefs.GetIndexForFields . . . 514
14.43.10 TIndexDefs.Update . . . 514
14.43.11 TIndexDefs.Items . . . 514
14.44 TIntegerField . . . 514
14.44.1 Description . . . 514
14.45 TLargeintField . . . 514
14.45.1 Description . . . 514
14.45.2 Method overview . . . 515
14.45.3 Property overview . . . 515
14.45.4 TLargeintField.Create . . . 515
14.45.5 TLargeintField.CheckRange . . . 515
14.45.6 TLargeintField.Value . . . 515
14.45.7 TLargeintField.MaxValue . . . 516
14.45.8 TLargeintField.MinValue . . . 516
14.46 TLongintField . . . 516

14.46.1 Description . . . 516
14.46.2 Method overview . . . 517
14.46.3 Property overview . . . 517
14.46.4 TLongintField.Create . . . 517
14.46.5 TLongintField.CheckRange . . . 517
14.46.6 TLongintField.Value . . . 517
14.46.7 TLongintField.MaxValue . . . 518
14.46.8 TLongintField.MinValue . . . 518
14.47TLongWordField . . . 518
14.47.1 Description . . . 518
14.47.2 Method overview . . . 518
14.47.3 Property overview . . . 518
14.47.4 TLongWordField.Create . . . 519
14.47.5 TLongWordField.CheckRange . . . 519
14.47.6 TLongWordField.Value . . . 519
14.47.7 TLongWordField.MaxValue . . . 519
14.47.8 TLongWordField.MinValue . . . 520
14.48TLookupList . . . 520
14.48.1 Description . . . 520
14.48.2 Method overview . . . 520
14.48.3 TLookupList.Create . . . 520
14.48.4 TLookupList.Destroy . . . 520
14.48.5 TLookupList.Add . . . 521
14.48.6 TLookupList.Clear . . . 521
14.48.7 TLookupList.FirstKeyByValue . . . 521
14.48.8 TLookupList.ValueOfKey . . . 521
14.48.9 TLookupList.ValuesToStrings . . . 522
14.49TMasterDataLink . . . 522
14.49.1 Description . . . 522
14.49.2 Method overview . . . 522
14.49.3 Property overview . . . 522
14.49.4 TMasterDataLink.Create . . . 522
14.49.5 TMasterDataLink.Destroy . . . 523
14.49.6 TMasterDataLink.FieldNames . . . 523
14.49.7 TMasterDataLink.Fields . . . 523
14.49.8 TMasterDataLink.OnMasterChange . . . 523
14.49.9 TMasterDataLink.OnMasterDisable . . . 524
14.50TMasterParamsDataLink . . . 524
14.50.1 Description . . . 524
14.50.2 Method overview . . . 524

14.50.3 Property overview . . . 524
14.50.4 TMasterParamsDataLink.Create . . . 524
14.50.5 TMasterParamsDataLink.RefreshParamNames . . . 525
14.50.6 TMasterParamsDataLink.CopyParamsFromMaster . . . 525
14.50.7 TMasterParamsDataLink.Params . . . 525
14.51TMemoField . . . 525
14.51.1 Description . . . 525
14.51.2 Method overview . . . 526
14.51.3 Property overview . . . 526
14.51.4 TMemoField.Create . . . 526
14.51.5 TMemoField.CodePage . . . 526
14.51.6 TMemoField.Transliterate . . . 526
14.52TNamedItem . . . 527
14.52.1 Description . . . 527
14.52.2 Property overview . . . 527
14.52.3 TNamedItem.DisplayName . . . 527
14.52.4 TNamedItem.Name . . . 527
14.53TNumericField . . . 527
14.53.1 Description . . . 527
14.53.2 Method overview . . . 528
14.53.3 Property overview . . . 528
14.53.4 TNumericField.Create . . . 528
14.53.5 TNumericField.Alignment . . . 528
14.53.6 TNumericField.DisplayFormat . . . 528
14.53.7 TNumericField.EditFormat . . . 529
14.54TObjectField . . . 529
14.54.1 Property overview . . . 529
14.54.2 TObjectField.FieldCount . . . 529
14.54.3 TObjectField.Fields . . . 529
14.54.4 TObjectField.FieldValues . . . 529
14.54.5 TObjectField.UnNamed . . . 530
14.54.6 TObjectField.ObjectType . . . 530
14.55TParam . . . 530
14.55.1 Description . . . 530
14.55.2 Method overview . . . 530
14.55.3 Property overview . . . 531
14.55.4 TParam.Create . . . 531
14.55.5 TParam.Assign . . . 532
14.55.6 TParam.AssignField . . . 532
14.55.7 TParam.AssignToField . . . 532

14.55.8 TParam.AssignFieldValue . . . 532
14.55.9 TParam.AssignFromField . . . 533
14.55.10TParam.Clear . . . 533
14.55.11TParam.GetData . . . 533
14.55.12TParam.GetDataSize . . . 533
14.55.13TParam.LoadFromFile . . . 534
14.55.14TParam.LoadFromStream . . . 534
14.55.15TParam.SetBlobData . . . 534
14.55.16TParam.SetData . . . 534
14.55.17TParam.AsBCD . . . 535
14.55.18TParam.AsBlob . . . 535
14.55.19TParam.AsBoolean . . . 535
14.55.20TParam.AsByte . . . 536
14.55.21TParam.AsBytes . . . 536
14.55.22TParam.AsCurrency . . . 536
14.55.23TParam.AsDate . . . 536
14.55.24TParam.AsDateTime . . . 537
14.55.25TParam.AsFloat . . . 537
14.55.26TParam.AsInteger . . . 537
14.55.27TParam.AsLargeInt . . . 537
14.55.28TParam.AsLongWord . . . 538
14.55.29TParam.AsMemo . . . 538
14.55.30TParam.AsShortInt . . . 538
14.55.31TParam.AsSingle . . . 538
14.55.32TParam.AsSmallInt . . . 538
14.55.33TParam.AsString . . . 539
14.55.34TParam.AsAnsiString . . . 539
14.55.35TParam.AsUTF8String . . . 539
14.55.36TParam.AsUnicodeString . . . 540
14.55.37TParam.AsTime . . . 540
14.55.38TParam.AsWord . . . 540
14.55.39TParam.AsFMTBCD . . . 540
14.55.40TParam.Bound . . . 541
14.55.41TParam.Dataset . . . 541
14.55.42TParam.IsNull . . . 541
14.55.43TParam.NativeStr . . . 541
14.55.44TParam.Text . . . 542
14.55.45TParam.AsWideString . . . 542
14.55.46TParam.DataType . . . 542
14.55.47TParam.Name . . . 542

14.55.48TParam.NumericScale . . . 543
14.55.49TParam.ParamType . . . 543
14.55.50TParam.Precision . . . 543
14.55.51TParam.Size . . . 544
14.55.52TParam.Value . . . 544
14.56TParams . . . 544
14.56.1 Description . . . 544
14.56.2 Method overview . . . 545
14.56.3 Property overview . . . 545
14.56.4 TParams.Create . . . 545
14.56.5 TParams.AddParam . . . 545
14.56.6 TParams.AssignValues . . . 546
14.56.7 TParams.CreateParam . . . 546
14.56.8 TParams.FindParam . . . 546
14.56.9 TParams.GetParamList . . . 546
14.56.10TParams.IsEqual . . . 547
14.56.11TParams.GetEnumerator . . . 547
14.56.12TParams.ParamByName . . . 547
14.56.13TParams.ParseSQL . . . 547
14.56.14TParams.RemoveParam . . . 548
14.56.15TParams.CopyParamValuesFromDataset . . . 549
14.56.16TParams.Dataset . . . 549
14.56.17TParams.Items . . . 549
14.56.18TParams.ParamValues . . . 549
14.57TParamsEnumerator . . . 550
14.57.1 Description . . . 550
14.57.2 Method overview . . . 550
14.57.3 Property overview . . . 550
14.57.4 TParamsEnumerator.Create . . . 550
14.57.5 TParamsEnumerator.MoveNext . . . 550
14.57.6 TParamsEnumerator.Current . . . 551
14.58TShortintField . . . 551
14.58.1 Description . . . 551
14.58.2 Method overview . . . 551
14.58.3 TShortintField.Create . . . 551
14.59TSingleField . . . 551
14.59.1 Method overview . . . 551
14.59.2 Property overview . . . 552
14.59.3 TSingleField.Create . . . 552
14.59.4 TSingleField.CheckRange . . . 552

14.59.5 TSingleField.Value . . . 552
14.59.6 TSingleField.Currency . . . 552
14.59.7 TSingleField.MaxValue . . . 552
14.59.8 TSingleField.MinValue . . . 552
14.59.9 TSingleField.Precision . . . 553
14.60TSmallintField . . . 553
14.60.1 Description . . . 553
14.60.2 Method overview . . . 553
14.60.3 TSmallintField.Create . . . 553
14.61TStringField . . . 553
14.61.1 Description . . . 553
14.61.2 Method overview . . . 554
14.61.3 Property overview . . . 554
14.61.4 TStringField.Create . . . 554
14.61.5 TStringField.SetFieldType . . . 554
14.61.6 TStringField.CodePage . . . 554
14.61.7 TStringField.FixedChar . . . 555
14.61.8 TStringField.Transliterate . . . 555
14.61.9 TStringField.Value . . . 555
14.61.10TStringField.EditMask . . . 555
14.61.11TStringField.Size . . . 556
14.62TTimeField . . . 556
14.62.1 Description . . . 556
14.62.2 Method overview . . . 556
14.62.3 TTimeField.Create . . . 556
14.63TVarBytesField . . . 557
14.63.1 Description . . . 557
14.63.2 Method overview . . . 557
14.63.3 TVarBytesField.Create . . . 557
14.64TVariantField . . . 557
14.64.1 Description . . . 557
14.64.2 Method overview . . . 557
14.64.3 TVariantField.Create . . . 558
14.65TWideMemoField . . . 558
14.65.1 Description . . . 558
14.65.2 Method overview . . . 558
14.65.3 Property overview . . . 558
14.65.4 TWideMemoField.Create . . . 558
14.65.5 TWideMemoField.Value . . . 559
14.66TWideStringField . . . 559

14.66.1 Description . . . 559
14.66.2 Method overview . . . 559
14.66.3 Property overview . . . 559
14.66.4 TWideStringField.Create . . . 559
14.66.5 TWideStringField.SetFieldType . . . 559
14.66.6 TWideStringField.Value . . . 560
14.67TWordField . . . 560
14.67.1 Description . . . 560
14.67.2 Method overview . . . 560
14.67.3 TWordField.Create . . . 560

15 Reference for unit 'dbugintf' 561
15.1 Used units . . . 561
15.2 Overview . . . 561
15.3 Writing a debug server. . . . 561
15.4 Constants, types and variables . . . 562
15.4.1 Resource strings . . . 562
15.4.2 Types . . . 562
15.4.3 Variables . . . 562
15.5 Procedures and functions . . . 563
15.5.1 FreeDebugClient . . . 563
15.5.2 GetDebuggingEnabled . . . 563
15.5.3 InitDebugClient . . . 563
15.5.4 SendBoolean . . . 564
15.5.5 SendDateTime . . . 564
15.5.6 SendDebug . . . 564
15.5.7 SendDebugEx . . . 564
15.5.8 SendDebugFmt . . . 565
15.5.9 SendDebugFmtEx . . . 565
15.5.10 SendInteger . . . 565
15.5.11 SendMethodEnter . . . 566
15.5.12 SendMethodExit . . . 566
15.5.13 SendPointer . . . 566
15.5.14 SendSeparator . . . 567
15.5.15 SetDebuggingEnabled . . . 567
15.5.16 StartDebugServer . . . 567

16 Reference for unit 'dbugmsg' 568
16.1 Used units . . . 568
16.2 Overview . . . 568
16.3 Constants, types and variables . . . 568

16.3.1 Constants . . . 568
16.3.2 Types . . . 569
16.4 Procedures and functions . . . 569
16.4.1 DebugMessageName . . . 569
16.4.2 ReadDebugMessageFromStream . . . 569
16.4.3 WriteDebugMessageToStream . . . 569
16.5 TDebugMessage . . . 570

17 Reference for unit 'eventlog' 571
17.1 Used units . . . 571
17.2 Overview . . . 571
17.3 Constants, types and variables . . . 571
17.3.1 Resource strings . . . 571
17.3.2 Types . . . 572
17.4 ELogError . . . 573
17.4.1 Description . . . 573
17.5 TEventLog . . . 573
17.5.1 Description . . . 573
17.5.2 Method overview . . . 573
17.5.3 Property overview . . . 574
17.5.4 TEventLog.Destroy . . . 574
17.5.5 TEventLog.EventTypeToString . . . 574
17.5.6 TEventLog.RegisterMessageFile . . . 574
17.5.7 TEventLog.UnRegisterMessageFile . . . 575
17.5.8 TEventLog.Pause . . . 576
17.5.9 TEventLog.Resume . . . 576
17.5.10 TEventLog.Log . . . 576
17.5.11 TEventLog.Warning . . . 576
17.5.12 TEventLog.Error . . . 577
17.5.13 TEventLog.Debug . . . 577
17.5.14 TEventLog.Info . . . 577
17.5.15 TEventLog.AppendContent . . . 577
17.5.16 TEventLog.Identification . . . 578
17.5.17 TEventLog.LogType . . . 578
17.5.18 TEventLog.Active . . . 578
17.5.19 TEventLog.RaiseExceptionOnError . . . 579
17.5.20 TEventLog.DefaultEventType . . . 579
17.5.21 TEventLog.FileName . . . 579
17.5.22 TEventLog.TimeStampFormat . . . 579
17.5.23 TEventLog.CustomLogType . . . 580

17.5.24 TEventLog.EventIDOffset . . . 580
17.5.25 TEventLog.OnGetCustomCategory . . . 580
17.5.26 TEventLog.OnGetCustomEventID . . . 581
17.5.27 TEventLog.OnGetCustomEvent . . . 581
17.5.28 TEventLog.OnLogMessage . . . 581
17.5.29 TEventLog.Paused . . . 581

18 Reference for unit 'ezcgi' 582
18.1 Used units . . . 582
18.2 Overview . . . 582
18.3 Constants, types and variables . . . 582
18.3.1 Constants . . . 582
18.4 ECGIException . . . 582
18.4.1 Description . . . 582
18.5 TEZcgi . . . 583
18.5.1 Description . . . 583
18.5.2 Method overview . . . 583
18.5.3 Property overview . . . 583
18.5.4 TEZcgi.Create . . . 583
18.5.5 TEZcgi.Destroy . . . 583
18.5.6 TEZcgi.Run . . . 584
18.5.7 TEZcgi.WriteContent . . . 584
18.5.8 TEZcgi.PutLine . . . 584
18.5.9 TEZcgi.GetValue . . . 585
18.5.10 TEZcgi.DoPost . . . 585
18.5.11 TEZcgi.DoGet . . . 585
18.5.12 TEZcgi.Values . . . 585
18.5.13 TEZcgi.Names . . . 586
18.5.14 TEZcgi.Variables . . . 586
18.5.15 TEZcgi.VariableCount . . . 587
18.5.16 TEZcgi.Name . . . 587
18.5.17 TEZcgi.Email . . . 587

19 Reference for unit 'fpjson' 588
19.1 Used units . . . 588
19.2 Overview . . . 588
19.3 Constants, types and variables . . . 590
19.3.1 Constants . . . 590
19.3.2 Types . . . 591
19.4 Procedures and functions . . . 595
19.4.1 CreateJSON . . . 595

19.4.2 CreateJSONArray . . . 595
19.4.3 CreateJSONObject . . . 595
19.4.4 GetJSON . . . 596
19.4.5 GetJSONInstanceType . . . 596
19.4.6 GetJSONParserHandler . . . 596
19.4.7 GetJSONStringParserHandler . . . 597
19.4.8 JSONStringToString . . . 597
19.4.9 JSONTypeName . . . 597
19.4.10 SetJSONInstanceType . . . 597
19.4.11 SetJSONParserHandler . . . 598
19.4.12 SetJSONStringParserHandler . . . 598
19.4.13 StringToJSONString . . . 598
19.5 TJSONEnum . . . 599
19.6 EJSON . . . 599
19.6.1 Description . . . 599
19.7 TBaseJSONEnumerator . . . 599
19.7.1 Description . . . 599
19.7.2 Method overview . . . 599
19.7.3 Property overview . . . 599
19.7.4 TBaseJSONEnumerator.GetCurrent . . . 599
19.7.5 TBaseJSONEnumerator.MoveNext . . . 600
19.7.6 TBaseJSONEnumerator.Current . . . 600
19.8 TJSONArray . . . 600
19.8.1 Description . . . 600
19.8.2 Method overview . . . 601
19.8.3 Property overview . . . 601
19.8.4 TJSONArray.Create . . . 601
19.8.5 TJSONArray.Destroy . . . 602
19.8.6 TJSONArray.JSONType . . . 602
19.8.7 TJSONArray.Clone . . . 602
19.8.8 TJSONArray.Iterate . . . 602
19.8.9 TJSONArray.IndexOf . . . 603
19.8.10 TJSONArray.GetEnumerator . . . 603
19.8.11 TJSONArray.Clear . . . 603
19.8.12 TJSONArray.Add . . . 603
19.8.13 TJSONArray.Delete . . . 604
19.8.14 TJSONArray.Exchange . . . 604
19.8.15 TJSONArray.Extract . . . 604
19.8.16 TJSONArray.Insert . . . 605
19.8.17 TJSONArray.Move . . . 605

19.8.18 TJSONArray.Remove . . . 605
19.8.19 TJSONArray.Sort . . . 606
19.8.20 TJSONArray.Items . . . 606
19.8.21 TJSONArray.Types . . . 606
19.8.22 TJSONArray.Nulls . . . 606
19.8.23 TJSONArray.Integers . . . 607
19.8.24 TJSONArray.Int64s . . . 607
19.8.25 TJSONArray.LargeInts . . . 608
19.8.26 TJSONArray.QWords . . . 608
19.8.27 TJSONArray.UnicodeStrings . . . 608
19.8.28 TJSONArray.Strings . . . 609
19.8.29 TJSONArray.Floats . . . 609
19.8.30 TJSONArray.Booleans . . . 610
19.8.31 TJSONArray.Arrays . . . 610
19.8.32 TJSONArray.Objects . . . 610
19.9 TJSONBoolean . . . 611
19.9.1 Description . . . 611
19.9.2 Method overview . . . 611
19.9.3 TJSONBoolean.Create . . . 611
19.9.4 TJSONBoolean.JSONType . . . 611
19.9.5 TJSONBoolean.Clear . . . 612
19.9.6 TJSONBoolean.Clone . . . 612
19.10TJSONData . . . 612
19.10.1 Description . . . 612
19.10.2 Method overview . . . 613
19.10.3 Property overview . . . 613
19.10.4 TJSONData.JSONType . . . 613
19.10.5 TJSONData.Create . . . 613
19.10.6 TJSONData.Clear . . . 614
19.10.7 TJSONData.DumpJSON . . . 614
19.10.8 TJSONData.GetEnumerator . . . 614
19.10.9 TJSONData.FindPath . . . 614
19.10.10TJSONData.GetPath . . . 616
19.10.11TJSONData.Clone . . . 617
19.10.12TJSONData.FormatJSON . . . 617
19.10.13TJSONData.CompressedJSON . . . 617
19.10.14TJSONData.Count . . . 618
19.10.15TJSONData.Items . . . 618
19.10.16TJSONData.Value . . . 618
19.10.17TJSONData.AsString . . . 618

19.10.18TJSONData.AsUnicodeString . . . 619
19.10.19TJSONData.AsInt64 . . . 619
19.10.20TJSONData.AsQWord . . . 619
19.10.21TJSONData.AsLargeInt . . . 620
19.10.22TJSONData.AsFloat . . . 620
19.10.23TJSONData.AsInteger . . . 620
19.10.24TJSONData.AsBoolean . . . 621
19.10.25TJSONData.IsNull . . . 621
19.10.26TJSONData.AsJSON . . . 621
19.11TJSONFloatNumber . . . 622
19.11.1 Description . . . 622
19.11.2 Method overview . . . 622
19.11.3 TJSONFloatNumber.Create . . . 622
19.11.4 TJSONFloatNumber.NumberType . . . 622
19.11.5 TJSONFloatNumber.Clear . . . 623
19.11.6 TJSONFloatNumber.Clone . . . 623
19.12TJSONInt64Number . . . 623
19.12.1 Description . . . 623
19.12.2 Method overview . . . 623
19.12.3 TJSONInt64Number.Create . . . 623
19.12.4 TJSONInt64Number.NumberType . . . 624
19.12.5 TJSONInt64Number.Clear . . . 624
19.12.6 TJSONInt64Number.Clone . . . 624
19.13TJSONIntegerNumber . . . 624
19.13.1 Description . . . 624
19.13.2 Method overview . . . 624
19.13.3 TJSONIntegerNumber.Create . . . 625
19.13.4 TJSONIntegerNumber.NumberType . . . 625
19.13.5 TJSONIntegerNumber.Clear . . . 625
19.13.6 TJSONIntegerNumber.Clone . . . 625
19.14TJSONNull . . . 625
19.14.1 Description . . . 625
19.14.2 Method overview . . . 626
19.14.3 TJSONNull.JSONType . . . 626
19.14.4 TJSONNull.Clear . . . 626
19.14.5 TJSONNull.Clone . . . 626
19.15TJSONNumber . . . 626
19.15.1 Description . . . 626
19.15.2 Method overview . . . 627
19.15.3 TJSONNumber.JSONType . . . 627

19.15.4 TJSONNumber.NumberType 627
19.16TJSONObject 627
19.16.1 Description 627
19.16.2 Method overview 628
19.16.3 Property overview 628
19.16.4 TJSONObject.Create 628
19.16.5 TJSONObject.Destroy 629
19.16.6 TJSONObject.JSONType 629
19.16.7 TJSONObject.Clone 629
19.16.8 TJSONObject.GetEnumerator 630
19.16.9 TJSONObject.Iterate 630
19.16.10TJSONObject.IndexOf 630
19.16.11TJSONObject.IndexOfName 630
19.16.12TJSONObject.Find 631
19.16.13TJSONObject.Get 631
19.16.14TJSONObject.Clear 632
19.16.15TJSONObject.Add 632
19.16.16TJSONObject.Delete 632
19.16.17TJSONObject.Remove 633
19.16.18TJSONObject.Extract 633
19.16.19TJSONObject.UnquotedMemberNames 633
19.16.20TJSONObject.Names 634
19.16.21TJSONObject.Elements 634
19.16.22TJSONObject.Types 634
19.16.23TJSONObject.Nulls 635
19.16.24TJSONObject.Floats 635
19.16.25TJSONObject.Integers 635
19.16.26TJSONObject.Int64s 635
19.16.27TJSONObject.QWords 636
19.16.28TJSONObject.LargeInts 636
19.16.29TJSONObject.UnicodeStrings 636
19.16.30TJSONObject.Strings 637
19.16.31TJSONObject.Booleans 637
19.16.32TJSONObject.Arrays 637
19.16.33TJSONObject.Objects 638
19.17TJSONQWordNumber 638
19.17.1 Description 638
19.17.2 Method overview 638
19.17.3 TJSONQWordNumber.Create 638
19.17.4 TJSONQWordNumber.NumberType 638

19.17.5 TJSONQWordNumber.Clear . . . 639
19.17.6 TJSONQWordNumber.Clone . . . 639
19.18TJSONString . . . 639
19.18.1 Description . . . 639
19.18.2 Method overview . . . 639
19.18.3 TJSONString.Create . . . 639
19.18.4 TJSONString.JSONType . . . 640
19.18.5 TJSONString.Clear . . . 640
19.18.6 TJSONString.Clone . . . 640

20 Reference for unit 'fpmimetypes' 641
20.1 Used units . . . 641
20.2 Overview . . . 641
20.3 Procedures and functions . . . 641
20.3.1 MimeTypes . . . 641
20.4 TFPMimeTypes . . . 642
20.4.1 Description . . . 642
20.4.2 Method overview . . . 642
20.4.3 TFPMimeTypes.Create . . . 642
20.4.4 TFPMimeTypes.Destroy . . . 642
20.4.5 TFPMimeTypes.Clear . . . 643
20.4.6 TFPMimeTypes.LoadKnownTypes . . . 643
20.4.7 TFPMimeTypes.GetNextExtension . . . 643
20.4.8 TFPMimeTypes.LoadFromStream . . . 643
20.4.9 TFPMimeTypes.LoadFromFile . . . 644
20.4.10 TFPMimeTypes.AddType . . . 644
20.4.11 TFPMimeTypes.GetMimeExtensions . . . 644
20.4.12 TFPMimeTypes.GetMimeType . . . 644
20.4.13 TFPMimeTypes.GetKnownMimeTypes . . . 645
20.4.14 TFPMimeTypes.GetKnownExtensions . . . 645
20.5 TMimeType . . . 645
20.5.1 Description . . . 645
20.5.2 Method overview . . . 645
20.5.3 Property overview . . . 645
20.5.4 TMimeType.Create . . . 646
20.5.5 TMimeType.MergeExtensions . . . 646
20.5.6 TMimeType.MimeType . . . 646
20.5.7 TMimeType.Extensions . . . 646

21 Reference for unit 'fptimer' 647
21.1 Used units . . . 647

21.2 Overview . . . 647
21.3 Constants, types and variables . . . 647
21.3.1 Types . . . 647
21.3.2 Variables . . . 647
21.4 TFPCustomTimer . . . 648
21.4.1 Description . . . 648
21.4.2 Method overview . . . 648
21.4.3 TFPCustomTimer.Create . . . 648
21.4.4 TFPCustomTimer.Destroy . . . 648
21.4.5 TFPCustomTimer.StartTimer . . . 649
21.4.6 TFPCustomTimer.StopTimer . . . 649
21.5 TFPTimer . . . 649
21.5.1 Description . . . 649
21.5.2 Property overview . . . 649
21.5.3 TFPTimer.Enabled . . . 649
21.5.4 TFPTimer.Interval . . . 650
21.5.5 TFPTimer.UseTimerThread . . . 650
21.5.6 TFPTimer.OnTimer . . . 650
21.5.7 TFPTimer.OnStartTimer . . . 650
21.5.8 TFPTimer.OnStopTimer . . . 651
21.6 TFPTimerDriver . . . 651
21.6.1 Description . . . 651
21.6.2 Method overview . . . 651
21.6.3 Property overview . . . 651
21.6.4 TFPTimerDriver.Create . . . 651
21.6.5 TFPTimerDriver.StartTimer . . . 651
21.6.6 TFPTimerDriver.StopTimer . . . 652
21.6.7 TFPTimerDriver.Timer . . . 652
21.6.8 TFPTimerDriver.TimerStarted . . . 652

22 Reference for unit 'gettext' 653
22.1 Used units . . . 653
22.2 Overview . . . 653
22.3 Constants, types and variables . . . 653
22.3.1 Constants . . . 653
22.3.2 Types . . . 653
22.4 Procedures and functions . . . 654
22.4.1 GetLanguageIDs . . . 654
22.4.2 TranslateResourceStrings . . . 654
22.4.3 TranslateUnitResourceStrings . . . 655

22.5 TMOFileHeader . . . 655
22.6 TMOStringInfo . . . 655
22.7 EMOFileError . . . 655
22.7.1 Description . . . 655
22.8 TMOFile . . . 656
22.8.1 Description . . . 656
22.8.2 Method overview . . . 656
22.8.3 TMOFile.Create . . . 656
22.8.4 TMOFile.Destroy . . . 656
22.8.5 TMOFile.Translate . . . 656

23 Reference for unit 'IBConnection' 658
23.1 Used units . . . 658
23.2 Constants, types and variables . . . 658
23.2.1 Constants . . . 658
23.2.2 Types . . . 658
23.3 TDatabaseInfo . . . 659
23.4 EIBDatabaseError . . . 659
23.4.1 Description . . . 659
23.4.2 Property overview . . . 659
23.4.3 EIBDatabaseError.StatusVector . . . 659
23.4.4 EIBDatabaseError.GDSErrorCode . . . 659
23.5 TIBConnection . . . 660
23.5.1 Description . . . 660
23.5.2 Method overview . . . 661
23.5.3 Property overview . . . 661
23.5.4 TIBConnection.Create . . . 661
23.5.5 TIBConnection.GetConnectionInfo . . . 661
23.5.6 TIBConnection.CreateDB . . . 662
23.5.7 TIBConnection.DropDB . . . 662
23.5.8 TIBConnection.BlobSegmentSize . . . 662
23.5.9 TIBConnection.ODSMajorVersion . . . 663
23.5.10 TIBConnection.DatabaseName . . . 663
23.5.11 TIBConnection.Dialect . . . 663
23.5.12 TIBConnection.CheckTransactionParams . . . 664
23.5.13 TIBConnection.KeepConnection . . . 664
23.5.14 TIBConnection.LoginPrompt . . . 664
23.5.15 TIBConnection.Params . . . 664
23.5.16 TIBConnection.OnLogin . . . 665
23.5.17 TIBConnection.Port . . . 665

23.5.18 TIBConnection.UseConnectionCharSetIfNone . . . 665
23.5.19 TIBConnection.WireCompression . . . 665
23.6 TIBConnectionDef . . . 666
23.6.1 Description . . . 666
23.6.2 Method overview . . . 666
23.6.3 TIBConnectionDef.TypeName . . . 666
23.6.4 TIBConnectionDef.ConnectionClass . . . 666
23.6.5 TIBConnectionDef.Description . . . 666
23.6.6 TIBConnectionDef.DefaultLibraryName . . . 667
23.6.7 TIBConnectionDef.LoadFunction . . . 667
23.6.8 TIBConnectionDef.UnLoadFunction . . . 667
23.6.9 TIBConnectionDef.LoadedLibraryName . . . 667
23.7 TIBCursor . . . 667
23.7.1 Description . . . 667
23.8 TIBTrans . . . 668
23.8.1 Description . . . 668

24 Reference for unit 'idea' 669
24.1 Used units . . . 669
24.2 Overview . . . 669
24.3 Constants, types and variables . . . 669
24.3.1 Constants . . . 669
24.3.2 Types . . . 670
24.4 Procedures and functions . . . 670
24.4.1 CipherIdea . . . 670
24.4.2 DeKeyIdea . . . 670
24.4.3 EnKeyIdea . . . 671
24.5 EIDEAError . . . 671
24.5.1 Description . . . 671
24.6 TIDEADeCryptStream . . . 671
24.6.1 Description . . . 671
24.6.2 Method overview . . . 671
24.6.3 TIDEADeCryptStream.Create . . . 672
24.6.4 TIDEADeCryptStream.Read . . . 672
24.6.5 TIDEADeCryptStream.Seek . . . 672
24.7 TIDEAEncryptStream . . . 673
24.7.1 Description . . . 673
24.7.2 Method overview . . . 673
24.7.3 TIDEAEncryptStream.Create . . . 673
24.7.4 TIDEAEncryptStream.Destroy . . . 673

24.7.5 TIDEAEncryptStream.Write . . . 674
24.7.6 TIDEAEncryptStream.Seek . . . 674
24.7.7 TIDEAEncryptStream.Flush . . . 674
24.8 TIDEAStream . . . 674
24.8.1 Description . . . 674
24.8.2 Method overview . . . 675
24.8.3 Property overview . . . 675
24.8.4 TIDEAStream.Create . . . 675
24.8.5 TIDEAStream.Key . . . 675

25 Reference for unit 'inicol' 676
25.1 Used units . . . 676
25.2 Overview . . . 676
25.3 Constants, types and variables . . . 676
25.3.1 Constants . . . 676
25.4 EIniCol . . . 677
25.4.1 Description . . . 677
25.5 TIniCollection . . . 677
25.5.1 Description . . . 677
25.5.2 Method overview . . . 677
25.5.3 Property overview . . . 677
25.5.4 TIniCollection.Load . . . 677
25.5.5 TIniCollection.Save . . . 678
25.5.6 TIniCollection.SaveToIni . . . 678
25.5.7 TIniCollection.SaveToFile . . . 678
25.5.8 TIniCollection.LoadFromIni . . . 679
25.5.9 TIniCollection.LoadFromFile . . . 679
25.5.10 TIniCollection.Prefix . . . 679
25.5.11 TIniCollection.SectionPrefix . . . 680
25.5.12 TIniCollection.FileName . . . 680
25.5.13 TIniCollection.GlobalSection . . . 680
25.6 TIniCollectionItem . . . 681
25.6.1 Description . . . 681
25.6.2 Method overview . . . 681
25.6.3 Property overview . . . 681
25.6.4 TIniCollectionItem.SaveToIni . . . 681
25.6.5 TIniCollectionItem.LoadFromIni . . . 681
25.6.6 TIniCollectionItem.SaveToFile . . . 682
25.6.7 TIniCollectionItem.LoadFromFile . . . 682
25.6.8 TIniCollectionItem.SectionName . . . 682

25.7 TNamedIniCollection . . . 683
25.7.1 Description . . . 683
25.7.2 Method overview . . . 683
25.7.3 Property overview . . . 683
25.7.4 TNamedIniCollection.IndexOfUserData . . . 683
25.7.5 TNamedIniCollection.IndexOfName . . . 683
25.7.6 TNamedIniCollection.FindByName . . . 684
25.7.7 TNamedIniCollection.FindByUserData . . . 684
25.7.8 TNamedIniCollection.NamedItems . . . 684
25.8 TNamedIniCollectionItem . . . 684
25.8.1 Description . . . 684
25.8.2 Property overview . . . 684
25.8.3 TNamedIniCollectionItem.UserData . . . 685
25.8.4 TNamedIniCollectionItem.Name . . . 685

26 Reference for unit 'IniFiles' 686
26.1 Used units . . . 686
26.2 Overview . . . 686
26.3 Constants, types and variables . . . 686
26.3.1 Types . . . 686
26.4 TCustomIniFile . . . 688
26.4.1 Description . . . 688
26.4.2 Method overview . . . 688
26.4.3 Property overview . . . 689
26.4.4 TCustomIniFile.Create . . . 689
26.4.5 TCustomIniFile.Destroy . . . 690
26.4.6 TCustomIniFile.SetBoolStringValues . . . 690
26.4.7 TCustomIniFile.SectionExists . . . 690
26.4.8 TCustomIniFile.ReadString . . . 690
26.4.9 TCustomIniFile.WriteString . . . 691
26.4.10 TCustomIniFile.ReadInteger . . . 691
26.4.11 TCustomIniFile.WriteInteger . . . 691
26.4.12 TCustomIniFile.ReadInt64 . . . 691
26.4.13 TCustomIniFile.WriteInt64 . . . 692
26.4.14 TCustomIniFile.ReadBool . . . 692
26.4.15 TCustomIniFile.WriteBool . . . 692
26.4.16 TCustomIniFile.ReadDate . . . 693
26.4.17 TCustomIniFile.ReadDateTime . . . 693
26.4.18 TCustomIniFile.ReadFloat . . . 693
26.4.19 TCustomIniFile.ReadTime . . . 693

26.4.20 TCustomIniFile.ReadBinaryStream . . . 694
26.4.21 TCustomIniFile.WriteDate . . . 694
26.4.22 TCustomIniFile.WriteDateTime . . . 694
26.4.23 TCustomIniFile.WriteFloat . . . 695
26.4.24 TCustomIniFile.WriteTime . . . 695
26.4.25 TCustomIniFile.WriteBinaryStream . . . 695
26.4.26 TCustomIniFile.ReadSection . . . 695
26.4.27 TCustomIniFile.ReadSections . . . 696
26.4.28 TCustomIniFile.ReadSectionValues . . . 696
26.4.29 TCustomIniFile.EraseSection . . . 696
26.4.30 TCustomIniFile.DeleteKey . . . 697
26.4.31 TCustomIniFile.UpdateFile . . . 697
26.4.32 TCustomIniFile.ValueExists . . . 697
26.4.33 TCustomIniFile.Encoding . . . 697
26.4.34 TCustomIniFile.FileName . . . 698
26.4.35 TCustomIniFile.Options . . . 698
26.4.36 TCustomIniFile.EscapeLineFeeds . . . 698
26.4.37 TCustomIniFile.CaseSensitive . . . 698
26.4.38 TCustomIniFile.StripQuotes . . . 699
26.4.39 TCustomIniFile.FormatSettingsActive . . . 699
26.4.40 TCustomIniFile.BoolTrueStrings . . . 699
26.4.41 TCustomIniFile.BoolFalseStrings . . . 700
26.4.42 TCustomIniFile.OwnsEncoding . . . 700
26.5 THashedStringList . . . 700
26.5.1 Description . . . 700
26.5.2 Method overview . . . 700
26.5.3 THashedStringList.Destroy . . . 700
26.5.4 THashedStringList.IndexOf . . . 701
26.5.5 THashedStringList.IndexOfName . . . 701
26.6 TIniFile . . . 701
26.6.1 Description . . . 701
26.6.2 Method overview . . . 701
26.6.3 Property overview . . . 702
26.6.4 TIniFile.Create . . . 702
26.6.5 TIniFile.Destroy . . . 702
26.6.6 TIniFile.ReadString . . . 702
26.6.7 TIniFile.WriteString . . . 703
26.6.8 TIniFile.ReadSection . . . 703
26.6.9 TIniFile.ReadSectionRaw . . . 703
26.6.10 TIniFile.ReadSections . . . 703

26.6.11 TIniFile.ReadSectionValues . . . 704
26.6.12 TIniFile.EraseSection . . . 704
26.6.13 TIniFile.DeleteKey . . . 704
26.6.14 TIniFile.UpdateFile . . . 704
26.6.15 TIniFile.Stream . . . 705
26.6.16 TIniFile.CacheUpdates . . . 705
26.6.17 TIniFile.WriteBOM . . . 705
26.7 TIniFileKey . . . 706
26.7.1 Description . . . 706
26.7.2 Method overview . . . 706
26.7.3 Property overview . . . 706
26.7.4 TIniFileKey.Create . . . 706
26.7.5 TIniFileKey.Ident . . . 706
26.7.6 TIniFileKey.Value . . . 706
26.8 TIniFileKeyList . . . 707
26.8.1 Description . . . 707
26.8.2 Method overview . . . 707
26.8.3 Property overview . . . 707
26.8.4 TIniFileKeyList.Destroy . . . 707
26.8.5 TIniFileKeyList.Clear . . . 707
26.8.6 TIniFileKeyList.Items . . . 707
26.9 TIniFileSection . . . 708
26.9.1 Description . . . 708
26.9.2 Method overview . . . 708
26.9.3 Property overview . . . 708
26.9.4 TIniFileSection.Empty . . . 708
26.9.5 TIniFileSection.Create . . . 708
26.9.6 TIniFileSection.Destroy . . . 709
26.9.7 TIniFileSection.Name . . . 709
26.9.8 TIniFileSection.KeyList . . . 709
26.10TIniFileSectionList . . . 709
26.10.1 Description . . . 709
26.10.2 Method overview . . . 709
26.10.3 Property overview . . . 710
26.10.4 TIniFileSectionList.Destroy . . . 710
26.10.5 TIniFileSectionList.Clear . . . 710
26.10.6 TIniFileSectionList.Items . . . 710
26.11TMemIniFile . . . 710
26.11.1 Description . . . 710
26.11.2 Method overview . . . 711

26.11.3 TMemIniFile.Create . . . 711
26.11.4 TMemIniFile.Clear . . . 711
26.11.5 TMemIniFile.GetStrings . . . 711
26.11.6 TMemIniFile.Rename . . . 712
26.11.7 TMemIniFile.SetStrings . . . 712
26.12 TStringHash . . . 712
26.12.1 Description . . . 712
26.12.2 Method overview . . . 712
26.12.3 Property overview . . . 712
26.12.4 TStringHash.Create . . . 713
26.12.5 TStringHash.Destroy . . . 713
26.12.6 TStringHash.Add . . . 713
26.12.7 TStringHash.Clear . . . 713
26.12.8 TStringHash.Modify . . . 714
26.12.9 TStringHash.Remove . . . 714
26.12.10 TStringHash.ValueOf . . . 714
26.12.11 TStringHash.AddReplacesExisting . . . 714

27 Reference for unit 'iostream' 715
27.1 Used units . . . 715
27.2 Overview . . . 715
27.3 Constants, types and variables . . . 715
27.3.1 Types . . . 715
27.4 EIOStreamError . . . 716
27.4.1 Description . . . 716
27.5 TIOStream . . . 716
27.5.1 Description . . . 716
27.5.2 Method overview . . . 716
27.5.3 TIOStream.Create . . . 716
27.5.4 TIOStream.Read . . . 716
27.5.5 TIOStream.Write . . . 717
27.5.6 TIOStream.Seek . . . 717

28 Reference for unit 'libtar' 718
28.1 Used units . . . 718
28.2 Overview . . . 718
28.3 Constants, types and variables . . . 718
28.3.1 Constants . . . 718
28.3.2 Types . . . 719
28.4 Procedures and functions . . . 721
28.4.1 ClearDirRec . . . 721

28.4.2 ConvertFilename . . . 721
28.4.3 FileTimeGMT . . . 721
28.4.4 PermissionString . . . 721
28.5 TTarArchive . . . 722
28.5.1 Description . . . 722
28.5.2 Method overview . . . 722
28.5.3 TTarArchive.Create . . . 722
28.5.4 TTarArchive.Destroy . . . 722
28.5.5 TTarArchive.Reset . . . 722
28.5.6 TTarArchive.FindNext . . . 723
28.5.7 TTarArchive.ReadFile . . . 723
28.5.8 TTarArchive.GetFilePos . . . 723
28.5.9 TTarArchive.SetFilePos . . . 724
28.6 TTarWriter . . . 724
28.6.1 Description . . . 724
28.6.2 Method overview . . . 724
28.6.3 Property overview . . . 724
28.6.4 TTarWriter.Create . . . 724
28.6.5 TTarWriter.Destroy . . . 725
28.6.6 TTarWriter.AddFile . . . 725
28.6.7 TTarWriter.AddStream . . . 725
28.6.8 TTarWriter.AddString . . . 726
28.6.9 TTarWriter.AddDir . . . 726
28.6.10 TTarWriter.AddSymbolicLink . . . 726
28.6.11 TTarWriter.AddLink . . . 727
28.6.12 TTarWriter.AddVolumeHeader . . . 727
28.6.13 TTarWriter.Finalize . . . 727
28.6.14 TTarWriter.Permissions . . . 727
28.6.15 TTarWriter.UID . . . 728
28.6.16 TTarWriter.GID . . . 728
28.6.17 TTarWriter.UserName . . . 728
28.6.18 TTarWriter.GroupName . . . 728
28.6.19 TTarWriter.Mode . . . 729
28.6.20 TTarWriter.Magic . . . 729

29 Reference for unit 'MaskUtils' 730
29.1 Used units . . . 730
29.2 Overview . . . 730
29.3 Constants, types and variables . . . 730
29.3.1 Types . . . 730

29.4 Procedures and functions . . . 732
29.4.1 FormatMaskInput . . . 732
29.4.2 FormatMaskText . . . 733
29.4.3 MaskDoFormatText . . . 733
29.5 TMaskUtils . . . 733
29.5.1 Description . . . 733
29.5.2 Method overview . . . 733
29.5.3 Property overview . . . 733
29.5.4 TMaskUtils.ValidateInput . . . 734
29.5.5 TMaskUtils.TryValidateInput . . . 734
29.5.6 TMaskUtils.Mask . . . 734
29.5.7 TMaskUtils.Value . . . 734
29.5.8 TMaskUtils.InputMask . . . 735

30 Reference for unit 'memds' 736
30.1 Used units . . . 736
30.2 Overview . . . 736
30.3 Constants, types and variables . . . 736
30.3.1 Constants . . . 736
30.4 MDSError . . . 737
30.4.1 Description . . . 737
30.5 TMemDataset . . . 738
30.5.1 Description . . . 738
30.5.2 Method overview . . . 739
30.5.3 Property overview . . . 740
30.5.4 TMemDataset.Create . . . 740
30.5.5 TMemDataset.Destroy . . . 740
30.5.6 TMemDataset.BookmarkValid . . . 741
30.5.7 TMemDataset.CompareBookmarks . . . 741
30.5.8 TMemDataset.CreateBlobStream . . . 741
30.5.9 TMemDataset.Locate . . . 742
30.5.10 TMemDataset.Lookup . . . 742
30.5.11 TMemDataset.CreateTable . . . 742
30.5.12 TMemDataset.DataSize . . . 743
30.5.13 TMemDataset.Clear . . . 743
30.5.14 TMemDataset.SaveToFile . . . 743
30.5.15 TMemDataset.SaveToStream . . . 744
30.5.16 TMemDataset.LoadFromStream . . . 744
30.5.17 TMemDataset.LoadFromFile . . . 744
30.5.18 TMemDataset.CopyFromDataset . . . 745

30.5.19 TMemDataset.FileModified . . . 746
30.5.20 TMemDataset.Filter . . . 746
30.5.21 TMemDataset.FileName . . . 746
30.5.22 TMemDataset.Filtered . . . 747
30.5.23 TMemDataset.Active . . . 747
30.5.24 TMemDataset.FieldDefs . . . 747
30.5.25 TMemDataset.BeforeOpen . . . 747
30.5.26 TMemDataset.AfterOpen . . . 747
30.5.27 TMemDataset.BeforeClose . . . 748
30.5.28 TMemDataset.AfterClose . . . 748
30.5.29 TMemDataset.BeforeInsert . . . 748
30.5.30 TMemDataset.AfterInsert . . . 748
30.5.31 TMemDataset.BeforeEdit . . . 748
30.5.32 TMemDataset.AfterEdit . . . 748
30.5.33 TMemDataset.BeforePost . . . 749
30.5.34 TMemDataset.AfterPost . . . 749
30.5.35 TMemDataset.BeforeCancel . . . 749
30.5.36 TMemDataset.AfterCancel . . . 749
30.5.37 TMemDataset.BeforeDelete . . . 749
30.5.38 TMemDataset.AfterDelete . . . 749
30.5.39 TMemDataset.BeforeScroll . . . 750
30.5.40 TMemDataset.AfterScroll . . . 750
30.5.41 TMemDataset.OnDeleteError . . . 750
30.5.42 TMemDataset.OnEditError . . . 750
30.5.43 TMemDataset.OnNewRecord . . . 750
30.5.44 TMemDataset.OnPostError . . . 751
30.5.45 TMemDataset.OnFilterRecord . . . 751

31 Reference for unit 'MSSQLConn' 752
31.1 Used units . . . 752
31.2 Overview . . . 752
31.3 Constants, types and variables . . . 752
31.3.1 Variables . . . 752
31.4 EMSSQLDatabaseError . . . 752
31.4.1 Description . . . 752
31.4.2 Property overview . . . 753
31.4.3 EMSSQLDatabaseError.DBErrorCode . . . 753
31.5 TMSSQLConnection . . . 753
31.5.1 Description . . . 753
31.5.2 Method overview . . . 754

31.5.3 Property overview . . . 754
31.5.4 TMSSQLConnection.Create . . . 754
31.5.5 TMSSQLConnection.GetConnectionInfo . . . 754
31.5.6 TMSSQLConnection.CreateDB . . . 754
31.5.7 TMSSQLConnection.DropDB . . . 755
31.5.8 TMSSQLConnection.Password . . . 755
31.5.9 TMSSQLConnection.Transaction . . . 755
31.5.10 TMSSQLConnection.UserName . . . 755
31.5.11 TMSSQLConnection.CharSet . . . 756
31.5.12 TMSSQLConnection.HostName . . . 756
31.5.13 TMSSQLConnection.Connected . . . 756
31.5.14 TMSSQLConnection.Role . . . 756
31.5.15 TMSSQLConnection.DatabaseName . . . 757
31.5.16 TMSSQLConnection.KeepConnection . . . 757
31.5.17 TMSSQLConnection.LoginPrompt . . . 757
31.5.18 TMSSQLConnection.Params . . . 757
31.5.19 TMSSQLConnection.OnLogin . . . 758
31.6 TMSSQLConnectionDef . . . 758
31.6.1 Description . . . 758
31.6.2 Method overview . . . 758
31.6.3 TMSSQLConnectionDef.TypeName . . . 758
31.6.4 TMSSQLConnectionDef.ConnectionClass . . . 758
31.6.5 TMSSQLConnectionDef.Description . . . 759
31.6.6 TMSSQLConnectionDef.DefaultLibraryName . . . 759
31.6.7 TMSSQLConnectionDef.LoadFunction . . . 759
31.6.8 TMSSQLConnectionDef.UnLoadFunction . . . 759
31.6.9 TMSSQLConnectionDef.LoadedLibraryName . . . 759
31.7 TSybaseConnection . . . 760
31.7.1 Description . . . 760
31.7.2 Method overview . . . 760
31.7.3 TSybaseConnection.Create . . . 760
31.8 TSybaseConnectionDef . . . 760
31.8.1 Description . . . 760
31.8.2 Method overview . . . 760
31.8.3 TSybaseConnectionDef.TypeName . . . 760
31.8.4 TSybaseConnectionDef.ConnectionClass . . . 761
31.8.5 TSybaseConnectionDef.Description . . . 761

32 Reference for unit 'nullstream' 762
32.1 Used units . . . 762

32.2 Overview . . . 762
32.3 ENullStreamError . . . 762
32.3.1 Description . . . 762
32.4 TNullStream . . . 762
32.4.1 Description . . . 762
32.4.2 Method overview . . . 763
32.4.3 TNullStream.Read . . . 763
32.4.4 TNullStream.Write . . . 763
32.4.5 TNullStream.Seek . . . 763
32.4.6 TNullStream.Create . . . 764

33 Reference for unit 'Pipes' 765
33.1 Used units . . . 765
33.2 Overview . . . 765
33.3 Constants, types and variables . . . 765
33.3.1 Constants . . . 765
33.4 Procedures and functions . . . 765
33.4.1 CreatePipeHandles . . . 765
33.4.2 CreatePipeStreams . . . 766
33.5 EPipeCreation . . . 766
33.5.1 Description . . . 766
33.6 EPipeError . . . 766
33.6.1 Description . . . 766
33.7 EPipeSeek . . . 766
33.7.1 Description . . . 766
33.8 TInputPipeStream . . . 767
33.8.1 Description . . . 767
33.8.2 Method overview . . . 767
33.8.3 Property overview . . . 767
33.8.4 TInputPipeStream.Destroy . . . 767
33.8.5 TInputPipeStream.Write . . . 767
33.8.6 TInputPipeStream.Seek . . . 768
33.8.7 TInputPipeStream.Read . . . 768
33.8.8 TInputPipeStream.NumBytesAvailable . . . 768
33.9 TOutputPipeStream . . . 769
33.9.1 Description . . . 769
33.9.2 Method overview . . . 769
33.9.3 Property overview . . . 769
33.9.4 TOutputPipeStream.Destroy . . . 769
33.9.5 TOutputPipeStream.Seek . . . 769

33.9.6 TOutputPipeStream.Read . . . 769
33.9.7 TOutputPipeStream.DontClose . . . 770

34 Reference for unit 'pooledmm' 771
34.1 Used units . . . 771
34.2 Overview . . . 771
34.3 Constants, types and variables . . . 771
34.3.1 Types . . . 771
34.4 TPooledMemManagerItem . . . 771
34.5 TNonFreePooledMemManager . . . 772
34.5.1 Description . . . 772
34.5.2 Method overview . . . 772
34.5.3 Property overview . . . 772
34.5.4 TNonFreePooledMemManager.Clear . . . 772
34.5.5 TNonFreePooledMemManager.Create . . . 772
34.5.6 TNonFreePooledMemManager.Destroy . . . 773
34.5.7 TNonFreePooledMemManager.NewItem . . . 773
34.5.8 TNonFreePooledMemManager.EnumerateItems . . . 773
34.5.9 TNonFreePooledMemManager.ItemSize . . . 773
34.6 TPooledMemManager . . . 774
34.6.1 Description . . . 774
34.6.2 Method overview . . . 774
34.6.3 Property overview . . . 774
34.6.4 TPooledMemManager.Clear . . . 774
34.6.5 TPooledMemManager.Create . . . 774
34.6.6 TPooledMemManager.Destroy . . . 775
34.6.7 TPooledMemManager.MinimumFreeCount . . . 775
34.6.8 TPooledMemManager.MaximumFreeCountRatio . . . 775
34.6.9 TPooledMemManager.Count . . . 775
34.6.10 TPooledMemManager.FreeCount . . . 776
34.6.11 TPooledMemManager.AllocatedCount . . . 776
34.6.12 TPooledMemManager.FreedCount . . . 776

35 Reference for unit 'process' 777
35.1 Used units . . . 777
35.2 Overview . . . 777
35.3 Constants, types and variables . . . 777
35.3.1 Types . . . 777
35.3.2 Variables . . . 781
35.4 Procedures and functions . . . 781
35.4.1 CommandToList . . . 781

35.4.2 DetectXTerm . . . 781
35.4.3 RunCommand . . . 782
35.4.4 RunCommandIndir . . . 782
35.5 EProcess . . . 783
35.5.1 Description . . . 783
35.6 TPROCESS . . . 783
35.6.1 Description . . . 783
35.6.2 Method overview . . . 783
35.6.3 Property overview . . . 784
35.6.4 TPROCESS.Create . . . 785
35.6.5 TPROCESS.Destroy . . . 785
35.6.6 TPROCESS.Execute . . . 785
35.6.7 TPROCESS.CloseInput . . . 786
35.6.8 TPROCESS.CloseOutput . . . 786
35.6.9 TPROCESS.CloseStderr . . . 786
35.6.10 TPROCESS.Resume . . . 786
35.6.11 TPROCESS.Suspend . . . 787
35.6.12 TPROCESS.Terminate . . . 787
35.6.13 TPROCESS.WaitOnExit . . . 787
35.6.14 TPROCESS.ReadInputStream . . . 788
35.6.15 TPROCESS.RunCommandLoop . . . 788
35.6.16 TPROCESS.WindowRect . . . 788
35.6.17 TPROCESS.Handle . . . 789
35.6.18 TPROCESS.ProcessHandle . . . 789
35.6.19 TPROCESS.ThreadHandle . . . 789
35.6.20 TPROCESS.ProcessID . . . 789
35.6.21 TPROCESS.ThreadID . . . 790
35.6.22 TPROCESS.Input . . . 790
35.6.23 TPROCESS.Output . . . 790
35.6.24 TPROCESS.Stderr . . . 791
35.6.25 TPROCESS.ExitStatus . . . 791
35.6.26 TPROCESS.ExitCode . . . 792
35.6.27 TPROCESS.InheritHandles . . . 792
35.6.28 TPROCESS.OnRunCommandEvent . . . 792
35.6.29 TPROCESS.RunCommandSleepTime . . . 792
35.6.30 TPROCESS.OnForkEvent . . . 793
35.6.31 TPROCESS.PipeBufferSize . . . 793
35.6.32 TPROCESS.Active . . . 793
35.6.33 TPROCESS.ApplicationName . . . 793
35.6.34 TPROCESS.CommandLine . . . 794

35.6.35 TPROCESS.Executable . . . 794
35.6.36 TPROCESS.Parameters . . . 795
35.6.37 TPROCESS.ConsoleTitle . . . 795
35.6.38 TPROCESS.CurrentDirectory . . . 796
35.6.39 TPROCESS.Desktop . . . 796
35.6.40 TPROCESS.Environment . . . 796
35.6.41 TPROCESS.Options . . . 797
35.6.42 TPROCESS.Priority . . . 797
35.6.43 TPROCESS.StartupOptions . . . 798
35.6.44 TPROCESS.Running . . . 798
35.6.45 TPROCESS.ShowWindow . . . 799
35.6.46 TPROCESS.WindowColumns . . . 799
35.6.47 TPROCESS.WindowHeight . . . 799
35.6.48 TPROCESS.WindowLeft . . . 800
35.6.49 TPROCESS.WindowRows . . . 800
35.6.50 TPROCESS.WindowTop . . . 800
35.6.51 TPROCESS.WindowWidth . . . 801
35.6.52 TPROCESS.FillAttribute . . . 801
35.6.53 TPROCESS.XTermProgram . . . 801

36 Reference for unit 'RttiUtils' 802
36.1 Used units . . . 802
36.2 Overview . . . 802
36.3 Constants, types and variables . . . 802
36.3.1 Constants . . . 802
36.3.2 Types . . . 802
36.3.3 Variables . . . 803
36.4 Procedures and functions . . . 804
36.4.1 CreateStoredItem . . . 804
36.4.2 ParseStoredItem . . . 804
36.4.3 UpdateStoredList . . . 804
36.5 TPropInfoList . . . 805
36.5.1 Description . . . 805
36.5.2 Method overview . . . 805
36.5.3 Property overview . . . 805
36.5.4 TPropInfoList.Create . . . 805
36.5.5 TPropInfoList.Destroy . . . 805
36.5.6 TPropInfoList.Contains . . . 806
36.5.7 TPropInfoList.Find . . . 806
36.5.8 TPropInfoList.Delete . . . 806

36.5.9 TPropInfoList.Intersect . . . 806
36.5.10 TPropInfoList.Count . . . 807
36.5.11 TPropInfoList.Items . . . 807
36.6 TPropsStorage . . . 807
36.6.1 Description . . . 807
36.6.2 Method overview . . . 807
36.6.3 Property overview . . . 808
36.6.4 TPropsStorage.StoreAnyProperty . . . 808
36.6.5 TPropsStorage.LoadAnyProperty . . . 808
36.6.6 TPropsStorage.StoreProperties . . . 808
36.6.7 TPropsStorage.LoadProperties . . . 809
36.6.8 TPropsStorage.LoadObjectsProps . . . 809
36.6.9 TPropsStorage.StoreObjectsProps . . . 809
36.6.10 TPropsStorage.Options . . . 810
36.6.11 TPropsStorage.AObject . . . 810
36.6.12 TPropsStorage.Prefix . . . 811
36.6.13 TPropsStorage.Section . . . 811
36.6.14 TPropsStorage.OnReadString . . . 811
36.6.15 TPropsStorage.OnWriteString . . . 811
36.6.16 TPropsStorage.OnEraseSection . . . 812

37 Reference for unit 'simpleipc' 813
37.1 Used units . . . 813
37.2 Overview . . . 813
37.3 Constants, types and variables . . . 814
37.3.1 Resource strings . . . 814
37.3.2 Constants . . . 814
37.3.3 Types . . . 814
37.3.4 Variables . . . 815
37.4 TMsgHeader . . . 816
37.5 EIPCError . . . 816
37.5.1 Description . . . 816
37.6 TIPCClientComm . . . 816
37.6.1 Description . . . 816
37.6.2 Method overview . . . 817
37.6.3 Property overview . . . 817
37.6.4 TIPCClientComm.Create . . . 817
37.6.5 TIPCClientComm.Connect . . . 817
37.6.6 TIPCClientComm.Disconnect . . . 817
37.6.7 TIPCClientComm.ServerRunning . . . 818

37.6.8 TIPCClientComm.SendMessage . . . 818
37.6.9 TIPCClientComm.Owner . . . 818
37.7 TIPCServerComm . . . 819
37.7.1 Description . . . 819
37.7.2 Method overview . . . 819
37.7.3 Property overview . . . 819
37.7.4 TIPCServerComm.Create . . . 819
37.7.5 TIPCServerComm.StartServer . . . 819
37.7.6 TIPCServerComm.StopServer . . . 820
37.7.7 TIPCServerComm.PeekMessage . . . 820
37.7.8 TIPCServerComm.ReadMessage . . . 820
37.7.9 TIPCServerComm.Owner . . . 821
37.7.10 TIPCServerComm.InstanceID . . . 821
37.8 TIPCServerMsg . . . 821
37.8.1 Description . . . 821
37.8.2 Method overview . . . 821
37.8.3 Property overview . . . 821
37.8.4 TIPCServerMsg.Create . . . 822
37.8.5 TIPCServerMsg.Destroy . . . 822
37.8.6 TIPCServerMsg.Stream . . . 822
37.8.7 TIPCServerMsg.MsgType . . . 822
37.8.8 TIPCServerMsg.OwnsStream . . . 823
37.8.9 TIPCServerMsg.StringMessage . . . 823
37.9 TIPCServerMsgQueue . . . 823
37.9.1 Description . . . 823
37.9.2 Method overview . . . 823
37.9.3 Property overview . . . 823
37.9.4 TIPCServerMsgQueue.Create . . . 824
37.9.5 TIPCServerMsgQueue.Destroy . . . 824
37.9.6 TIPCServerMsgQueue.Clear . . . 824
37.9.7 TIPCServerMsgQueue.Push . . . 824
37.9.8 TIPCServerMsgQueue.Pop . . . 825
37.9.9 TIPCServerMsgQueue.Count . . . 825
37.9.10 TIPCServerMsgQueue.MaxCount . . . 825
37.9.11 TIPCServerMsgQueue.MaxAction . . . 825
37.10TSimpleIPC . . . 826
37.10.1 Description . . . 826
37.10.2 Property overview . . . 826
37.10.3 TSimpleIPC.Active . . . 826
37.10.4 TSimpleIPC.ServerID . . . 826

37.11TSimpleIPCClient 827
37.11.1 Description 827
37.11.2 Method overview 827
37.11.3 Property overview 827
37.11.4 TSimpleIPCClient.Create 827
37.11.5 TSimpleIPCClient.Destroy 827
37.11.6 TSimpleIPCClient.Connect 828
37.11.7 TSimpleIPCClient.Disconnect 828
37.11.8 TSimpleIPCClient.ServerRunning 828
37.11.9 TSimpleIPCClient.SendMessage 829
37.11.10TSimpleIPCClient.SendStringMessage 829
37.11.11TSimpleIPCClient.SendStringMessageFmt 829
37.11.12TSimpleIPCClient.ServerInstance 829
37.12TSimpleIPCServer 830
37.12.1 Description 830
37.12.2 Method overview 830
37.12.3 Property overview 830
37.12.4 TSimpleIPCServer.Create 831
37.12.5 TSimpleIPCServer.Destroy 831
37.12.6 TSimpleIPCServer.StartServer 831
37.12.7 TSimpleIPCServer.StopServer 832
37.12.8 TSimpleIPCServer.PeekMessage 832
37.12.9 TSimpleIPCServer.ReadMessage 832
37.12.10TSimpleIPCServer.GetMessageData 832
37.12.11TSimpleIPCServer.StringMessage 833
37.12.12TSimpleIPCServer.Message 833
37.12.13TSimpleIPCServer.MsgType 833
37.12.14TSimpleIPCServer.MsgData 833
37.12.15TSimpleIPCServer.InstanceID 834
37.12.16TSimpleIPCServer.ThreadExecuting 834
37.12.17TSimpleIPCServer.ThreadError 834
37.12.18TSimpleIPCServer.Global 834
37.12.19TSimpleIPCServer.OnMessage 835
37.12.20TSimpleIPCServer.OnMessageQueued 835
37.12.21TSimpleIPCServer.OnMessageError 835
37.12.22TSimpleIPCServer.OnThreadError 835
37.12.23TSimpleIPCServer.MaxQueue 836
37.12.24TSimpleIPCServer.MaxAction 836
37.12.25TSimpleIPCServer.Threaded 836
37.12.26TSimpleIPCServer.ThreadTimeout 837

37.12.27TSimpleIPCServer.SynchronizeEvents . . . 837

38 Reference for unit 'singleinstance' 838
38.1 Used units . . . 838
38.2 Overview . . . 838
38.3 Constants, types and variables . . . 838
38.3.1 Types . . . 838
38.3.2 Variables . . . 839
38.4 ESingleInstance . . . 839
38.4.1 Description . . . 839
38.5 TBaseSingleInstance . . . 839
38.5.1 Description . . . 839
38.5.2 Method overview . . . 840
38.5.3 Property overview . . . 840
38.5.4 TBaseSingleInstance.Create . . . 840
38.5.5 TBaseSingleInstance.Destroy . . . 840
38.5.6 TBaseSingleInstance.Start . . . 840
38.5.7 TBaseSingleInstance.Stop . . . 841
38.5.8 TBaseSingleInstance.ServerCheckMessages . . . 841
38.5.9 TBaseSingleInstance.ClientPostParams . . . 841
38.5.10 TBaseSingleInstance.TimeOutMessages . . . 842
38.5.11 TBaseSingleInstance.TimeOutWaitForInstances . . . 842
38.5.12 TBaseSingleInstance.OnServerReceivedParams . . . 842
38.5.13 TBaseSingleInstance.StartResult . . . 842
38.5.14 TBaseSingleInstance.IsServer . . . 843
38.5.15 TBaseSingleInstance.IsClient . . . 843

39 Reference for unit 'SQLDB' 844
39.1 Used units . . . 844
39.2 Overview . . . 844
39.3 Using SQLDB to access databases. . . . 845
39.4 Using the universal TSQLConnector type. . . . 847
39.5 Retrieving Schema Information. . . . 848
39.6 Automatic generation of update SQL statements. . . . 848
39.7 Using parameters. . . . 849
39.8 Constants, types and variables . . . 850
39.8.1 Constants . . . 850
39.8.2 Types . . . 854
39.8.3 Variables . . . 857
39.9 Procedures and functions . . . 858
39.9.1 GetConnectionDef . . . 858

39.9.2 GetConnectionList . . . 858
39.9.3 RegisterConnection . . . 858
39.9.4 UnRegisterConnection . . . 859
39.10TSQLStatementInfo . . . 859
39.11ESQLDatabaseError . . . 859
39.11.1 Description . . . 859
39.11.2 Method overview . . . 859
39.11.3 ESQLDatabaseError.CreateFmt . . . 859
39.12TConnectionDef . . . 860
39.12.1 Description . . . 860
39.12.2 Method overview . . . 860
39.12.3 TConnectionDef.TypeName . . . 860
39.12.4 TConnectionDef.ConnectionClass . . . 861
39.12.5 TConnectionDef.Description . . . 861
39.12.6 TConnectionDef.DefaultLibraryName . . . 861
39.12.7 TConnectionDef.LoadFunction . . . 861
39.12.8 TConnectionDef.UnLoadFunction . . . 862
39.12.9 TConnectionDef.LoadedLibraryName . . . 862
39.12.10TConnectionDef.ApplyParams . . . 862
39.13TCustomSQLQuery . . . 862
39.13.1 Description . . . 862
39.13.2 Method overview . . . 863
39.13.3 Property overview . . . 863
39.13.4 TCustomSQLQuery.Create . . . 863
39.13.5 TCustomSQLQuery.Destroy . . . 863
39.13.6 TCustomSQLQuery.Prepare . . . 864
39.13.7 TCustomSQLQuery.UnPrepare . . . 864
39.13.8 TCustomSQLQuery.ExecSQL . . . 864
39.13.9 TCustomSQLQuery.SetSchemaInfo . . . 865
39.13.10TCustomSQLQuery.RowsAffected . . . 865
39.13.11TCustomSQLQuery.ParamByName . . . 865
39.13.12TCustomSQLQuery.MacroByName . . . 866
39.13.13TCustomSQLQuery.ApplyUpdates . . . 866
39.13.14TCustomSQLQuery.Post . . . 866
39.13.15TCustomSQLQuery.Delete . . . 866
39.13.16TCustomSQLQuery.Prepared . . . 867
39.13.17TCustomSQLQuery.SQLConnection . . . 867
39.13.18TCustomSQLQuery.SQLTransaction . . . 867
39.14TCustomSQLStatement . . . 867
39.14.1 Description . . . 867

39.14.2 Method overview . . . 868
39.14.3 Property overview . . . 868
39.14.4 TCustomSQLStatement.Create . . . 868
39.14.5 TCustomSQLStatement.Destroy . . . 868
39.14.6 TCustomSQLStatement.Prepare . . . 869
39.14.7 TCustomSQLStatement.Execute . . . 869
39.14.8 TCustomSQLStatement.Unprepare . . . 869
39.14.9 TCustomSQLStatement.ParamByName . . . 870
39.14.10TCustomSQLStatement.RowsAffected . . . 870
39.14.11TCustomSQLStatement.Prepared . . . 870
39.15TServerIndexDefs . . . 870
39.15.1 Description . . . 870
39.15.2 Method overview . . . 870
39.15.3 TServerIndexDefs.Create . . . 871
39.15.4 TServerIndexDefs.Update . . . 871
39.16TSQLConnection . . . 871
39.16.1 Description . . . 871
39.16.2 Method overview . . . 871
39.16.3 Property overview . . . 872
39.16.4 TSQLConnection.Create . . . 872
39.16.5 TSQLConnection.Destroy . . . 872
39.16.6 TSQLConnection.StartTransaction . . . 873
39.16.7 TSQLConnection.EndTransaction . . . 873
39.16.8 TSQLConnection.ExecuteDirect . . . 873
39.16.9 TSQLConnection.GetObjectNames . . . 874
39.16.10TSQLConnection.HasTable . . . 874
39.16.11TSQLConnection.GetTableNames . . . 874
39.16.12TSQLConnection.GetProcedureNames . . . 874
39.16.13TSQLConnection.GetFieldNames . . . 875
39.16.14TSQLConnection.GetSchemaNames . . . 875
39.16.15TSQLConnection.GetSequenceNames . . . 875
39.16.16TSQLConnection.GetConnectionInfo . . . 875
39.16.17TSQLConnection.GetStatementInfo . . . 876
39.16.18TSQLConnection.CreateDB . . . 876
39.16.19TSQLConnection.DropDB . . . 876
39.16.20TSQLConnection.GetNextValue . . . 876
39.16.21TSQLConnection.ConnOptions . . . 877
39.16.22TSQLConnection.Handle . . . 877
39.16.23TSQLConnection.FieldNameQuoteChars . . . 877
39.16.24TSQLConnection.Password . . . 878

39.16.25TSQLConnection.Transaction . . . 878
39.16.26TSQLConnection.UserName . . . 878
39.16.27TSQLConnection.CharSet . . . 879
39.16.28TSQLConnection.HostName . . . 879
39.16.29TSQLConnection.OnLog . . . 879
39.16.30TSQLConnection.LogEvents . . . 880
39.16.31TSQLConnection.Options . . . 880
39.16.32TSQLConnection.Role . . . 880
39.16.33TSQLConnection.Connected . . . 881
39.16.34TSQLConnection.DatabaseName . . . 881
39.16.35TSQLConnection.KeepConnection . . . 881
39.16.36TSQLConnection.LoginPrompt . . . 882
39.16.37TSQLConnection.Params . . . 882
39.16.38TSQLConnection.OnLogin . . . 882
39.17TSQLConnector . . . 882
39.17.1 Description . . . 882
39.17.2 Property overview . . . 883
39.17.3 TSQLConnector.ConnectorType . . . 883
39.17.4 TSQLConnector.Port . . . 883
39.18TSQLCursor . . . 883
39.18.1 Description . . . 883
39.19TSQLDBFieldDef . . . 884
39.19.1 Description . . . 884
39.19.2 Property overview . . . 884
39.19.3 TSQLDBFieldDef.SQLDBData . . . 884
39.20TSQLDBFieldDefs . . . 884
39.20.1 Description . . . 884
39.21TSQLDBParam . . . 884
39.21.1 Description . . . 884
39.21.2 Property overview . . . 884
39.21.3 TSQLDBParam.FieldDef . . . 885
39.21.4 TSQLDBParam.SQLDBData . . . 885
39.22TSQLDBParams . . . 885
39.22.1 Description . . . 885
39.23TSQLHandle . . . 885
39.23.1 Description . . . 885
39.24TSQLQuery . . . 886
39.24.1 Description . . . 886
39.24.2 Property overview . . . 888
39.24.3 TSQLQuery.SchemaType . . . 889

39.24.4 TSQLQuery.StatementType . . . 889
39.24.5 TSQLQuery.MaxIndexesCount . . . 889
39.24.6 TSQLQuery.FieldDefs . . . 889
39.24.7 TSQLQuery.Active . . . 890
39.24.8 TSQLQuery.AutoCalcFields . . . 890
39.24.9 TSQLQuery.Filter . . . 890
39.24.10TSQLQuery.Filtered . . . 890
39.24.11TSQLQuery.AfterCancel . . . 890
39.24.12TSQLQuery.AfterClose . . . 890
39.24.13TSQLQuery.AfterDelete . . . 890
39.24.14TSQLQuery.AfterEdit . . . 891
39.24.15TSQLQuery.AfterInsert . . . 891
39.24.16TSQLQuery.AfterOpen . . . 891
39.24.17TSQLQuery.AfterPost . . . 891
39.24.18TSQLQuery.AfterRefresh . . . 891
39.24.19TSQLQuery.AfterScroll . . . 891
39.24.20TSQLQuery.BeforeCancel . . . 891
39.24.21TSQLQuery.BeforeClose . . . 892
39.24.22TSQLQuery.BeforeDelete . . . 892
39.24.23TSQLQuery.BeforeEdit . . . 892
39.24.24TSQLQuery.BeforeInsert . . . 892
39.24.25TSQLQuery.BeforeOpen . . . 892
39.24.26TSQLQuery.BeforePost . . . 892
39.24.27TSQLQuery.BeforeRefresh . . . 892
39.24.28TSQLQuery.BeforeScroll . . . 893
39.24.29TSQLQuery.OnCalcFields . . . 893
39.24.30TSQLQuery.OnDeleteError . . . 893
39.24.31TSQLQuery.OnEditError . . . 893
39.24.32TSQLQuery.OnFilterRecord . . . 893
39.24.33TSQLQuery.OnNewRecord . . . 893
39.24.34TSQLQuery.OnPostError . . . 893
39.24.35TSQLQuery.Database . . . 894
39.24.36TSQLQuery.Transaction . . . 894
39.24.37TSQLQuery.ReadOnly . . . 894
39.24.38TSQLQuery.SQL . . . 894
39.24.39TSQLQuery.InsertSQL . . . 895
39.24.40TSQLQuery.UpdateSQL . . . 895
39.24.41TSQLQuery.DeleteSQL . . . 896
39.24.42TSQLQuery.RefreshSQL . . . 896
39.24.43TSQLQuery.IndexDefs . . . 896

39.24.44TSQLQuery.Options . . . 897
39.24.45TSQLQuery.Params . . . 897
39.24.46TSQLQuery.ParamCheck . . . 898
39.24.47TSQLQuery.Macros . . . 898
39.24.48TSQLQuery.MacroCheck . . . 898
39.24.49TSQLQuery.MacroChar . . . 898
39.24.50TSQLQuery.ParseSQL . . . 899
39.24.51TSQLQuery.UpdateMode . . . 899
39.24.52TSQLQuery.UsePrimaryKeyAsKey . . . 899
39.24.53TSQLQuery.DataSource . . . 900
39.24.54TSQLQuery.Sequence . . . 900
39.24.55TSQLQuery.ServerFilter . . . 901
39.24.56TSQLQuery.ServerFiltered . . . 901
39.24.57TSQLQuery.ServerIndexDefs . . . 901
39.25TSQLScript . . . 902
39.25.1 Description . . . 902
39.25.2 Method overview . . . 902
39.25.3 Property overview . . . 902
39.25.4 TSQLScript.Create . . . 902
39.25.5 TSQLScript.Destroy . . . 903
39.25.6 TSQLScript.Execute . . . 903
39.25.7 TSQLScript.ExecuteScript . . . 903
39.25.8 TSQLScript.Aborted . . . 903
39.25.9 TSQLScript.Line . . . 904
39.25.10TSQLScript.DataBase . . . 904
39.25.11TSQLScript.Transaction . . . 904
39.25.12TSQLScript.OnDirective . . . 904
39.25.13TSQLScript.AutoCommit . . . 905
39.25.14TSQLScript.UseDollarString . . . 905
39.25.15TSQLScript.DollarStrings . . . 905
39.25.16TSQLScript.Directives . . . 906
39.25.17TSQLScript.Defines . . . 906
39.25.18TSQLScript.Script . . . 906
39.25.19TSQLScript.Terminator . . . 907
39.25.20TSQLScript.CommentsinSQL . . . 907
39.25.21TSQLScript.UseSetTerm . . . 907
39.25.22TSQLScript.UseCommit . . . 908
39.25.23TSQLScript.UseDefines . . . 908
39.25.24TSQLScript.OnException . . . 909
39.26TSQLSequence . . . 909

39.26.1 Description . . . 909
39.26.2 Method overview . . . 909
39.26.3 Property overview . . . 909
39.26.4 TSQLSequence.Create . . . 909
39.26.5 TSQLSequence.Assign . . . 910
39.26.6 TSQLSequence.Apply . . . 910
39.26.7 TSQLSequence.GetNextValue . . . 910
39.26.8 TSQLSequence.FieldName . . . 910
39.26.9 TSQLSequence.SequenceName . . . 911
39.26.10TSQLSequence.IncrementBy . . . 911
39.26.11TSQLSequence.ApplyEvent . . . 911
39.27TSQLStatement . . . 911
39.27.1 Description . . . 911
39.27.2 Property overview . . . 912
39.27.3 TSQLStatement.Database . . . 912
39.27.4 TSQLStatement.DataSource . . . 912
39.27.5 TSQLStatement.ParamCheck . . . 912
39.27.6 TSQLStatement.Params . . . 913
39.27.7 TSQLStatement.MacroCheck . . . 913
39.27.8 TSQLStatement.Macros . . . 913
39.27.9 TSQLStatement.ParseSQL . . . 914
39.27.10TSQLStatement.SQL . . . 914
39.27.11TSQLStatement.Transaction . . . 914
39.28TSQLTransaction . . . 915
39.28.1 Description . . . 915
39.28.2 Method overview . . . 915
39.28.3 Property overview . . . 915
39.28.4 TSQLTransaction.Create . . . 915
39.28.5 TSQLTransaction.Destroy . . . 915
39.28.6 TSQLTransaction.Commit . . . 916
39.28.7 TSQLTransaction.CommitRetaining . . . 916
39.28.8 TSQLTransaction.Rollback . . . 916
39.28.9 TSQLTransaction.RollbackRetaining . . . 917
39.28.10TSQLTransaction.StartTransaction . . . 917
39.28.11TSQLTransaction.EndTransaction . . . 917
39.28.12TSQLTransaction.Handle . . . 918
39.28.13TSQLTransaction.SQLConnection . . . 918
39.28.14TSQLTransaction.Action . . . 918
39.28.15TSQLTransaction.Database . . . 918
39.28.16TSQLTransaction.Params . . . 919

39.28.17 TSQLTransaction.Options 919

40 Reference for unit 'SQLTypes' 920
40.1 Used units 920
40.2 Constants, types and variables 920
40.2.1 Types 920
40.3 TSqlObjectIdenfier 922
40.3.1 Description 922
40.3.2 Method overview 922
40.3.3 Property overview 922
40.3.4 TSqlObjectIdenfier.Create 922
40.3.5 TSqlObjectIdenfier.FullName 923
40.3.6 TSqlObjectIdenfier.SchemaName 923
40.3.7 TSqlObjectIdenfier.ObjectName 923
40.4 TSqlObjectIdentifierList 923
40.4.1 Method overview 923
40.4.2 Property overview 924
40.4.3 TSqlObjectIdentifierList.AddIdentifier 924
40.4.4 TSqlObjectIdentifierList.Identifiers 924

41 Reference for unit 'streamcoll' 925
41.1 Used units 925
41.2 Overview 925
41.3 Procedures and functions 925
41.3.1 ColReadBoolean 925
41.3.2 ColReadCurrency 926
41.3.3 ColReadDateTime 926
41.3.4 ColReadFloat 926
41.3.5 ColReadInteger 926
41.3.6 ColReadString 927
41.3.7 ColWriteBoolean 927
41.3.8 ColWriteCurrency 927
41.3.9 ColWriteDateTime 927
41.3.10 ColWriteFloat 928
41.3.11 ColWriteInteger 928
41.3.12 ColWriteString 928
41.4 EStreamColl 928
41.4.1 Description 928
41.5 TStreamCollection 928
41.5.1 Description 928
41.5.2 Method overview 929

41.5.3 Property overview . . . 929
41.5.4 TStreamCollection.LoadFromStream . . . 929
41.5.5 TStreamCollection.SaveToStream . . . 929
41.5.6 TStreamCollection.Streaming . . . 929
41.6 TStreamCollectionItem . . . 930
41.6.1 Description . . . 930

42 Reference for unit 'streamex' 931
42.1 Used units . . . 931
42.2 Overview . . . 931
42.3 Constants, types and variables . . . 931
42.3.1 Constants . . . 931
42.4 TBidirBinaryObjectReader . . . 932
42.4.1 Description . . . 932
42.4.2 Property overview . . . 932
42.4.3 TBidirBinaryObjectReader.Position . . . 932
42.5 TBidirBinaryObjectWriter . . . 932
42.5.1 Description . . . 932
42.5.2 Property overview . . . 932
42.5.3 TBidirBinaryObjectWriter.Position . . . 932
42.6 TDelphiReader . . . 933
42.6.1 Description . . . 933
42.6.2 Method overview . . . 933
42.6.3 Property overview . . . 933
42.6.4 TDelphiReader.GetDriver . . . 933
42.6.5 TDelphiReader.ReadStr . . . 933
42.6.6 TDelphiReader.Read . . . 933
42.6.7 TDelphiReader.Position . . . 934
42.7 TDelphiWriter . . . 934
42.7.1 Description . . . 934
42.7.2 Method overview . . . 934
42.7.3 Property overview . . . 934
42.7.4 TDelphiWriter.GetDriver . . . 934
42.7.5 TDelphiWriter.FlushBuffer . . . 935
42.7.6 TDelphiWriter.Write . . . 935
42.7.7 TDelphiWriter.WriteStr . . . 935
42.7.8 TDelphiWriter.WriteValue . . . 935
42.7.9 TDelphiWriter.Position . . . 935
42.8 TFileReader . . . 936
42.8.1 Description . . . 936

42.8.2 Method overview . . . 936
42.8.3 TFileReader.Create . . . 936
42.8.4 TFileReader.Destroy . . . 936
42.8.5 TFileReader.Reset . . . 937
42.8.6 TFileReader.Close . . . 937
42.8.7 TFileReader.ReadLine . . . 937
42.9 TStreamHelper . . . 937
42.9.1 Description . . . 937
42.9.2 Method overview . . . 938
42.9.3 TStreamHelper.ReadWordLE . . . 938
42.9.4 TStreamHelper.ReadDWordLE . . . 938
42.9.5 TStreamHelper.ReadQWordLE . . . 938
42.9.6 TStreamHelper.WriteWordLE . . . 939
42.9.7 TStreamHelper.WriteDWordLE . . . 939
42.9.8 TStreamHelper.WriteQWordLE . . . 939
42.9.9 TStreamHelper.ReadWordBE . . . 940
42.9.10 TStreamHelper.ReadDWordBE . . . 940
42.9.11 TStreamHelper.ReadQWordBE . . . 940
42.9.12 TStreamHelper.WriteWordBE . . . 940
42.9.13 TStreamHelper.WriteDWordBE . . . 941
42.9.14 TStreamHelper.WriteQWordBE . . . 941
42.9.15 TStreamHelper.ReadSingle . . . 941
42.9.16 TStreamHelper.ReadDouble . . . 941
42.9.17 TStreamHelper.WriteSingle . . . 942
42.9.18 TStreamHelper.WriteDouble . . . 942
42.10 TStreamReader . . . 942
42.10.1 Description . . . 942
42.10.2 Method overview . . . 942
42.10.3 Property overview . . . 942
42.10.4 TStreamReader.Create . . . 943
42.10.5 TStreamReader.Destroy . . . 943
42.10.6 TStreamReader.Reset . . . 943
42.10.7 TStreamReader.Close . . . 943
42.10.8 TStreamReader.ReadLine . . . 944
42.10.9 TStreamReader.BaseStream . . . 944
42.10.10 TStreamReader.OwnsStream . . . 944
42.11 TStreamWriter . . . 945
42.11.1 Method overview . . . 945
42.11.2 Property overview . . . 945
42.11.3 TStreamWriter.Create . . . 945

42.11.4 TStreamWriter.Destroy . . . 945
42.11.5 TStreamWriter.Close . . . 945
42.11.6 TStreamWriter.Flush . . . 945
42.11.7 TStreamWriter.OwnStream . . . 946
42.11.8 TStreamWriter.Write . . . 946
42.11.9 TStreamWriter.WriteLine . . . 946
42.11.10TStreamWriter.AutoFlush . . . 946
42.11.11TStreamWriter.NewLine . . . 947
42.11.12TStreamWriter.Encoding . . . 947
42.11.13TStreamWriter.BaseStream . . . 947
42.12TStringReader . . . 947
42.12.1 Description . . . 947
42.12.2 Method overview . . . 947
42.12.3 TStringReader.Create . . . 947
42.12.4 TStringReader.Destroy . . . 948
42.12.5 TStringReader.Reset . . . 948
42.12.6 TStringReader.Close . . . 948
42.12.7 TStringReader.ReadLine . . . 948
42.13TStringWriter . . . 949
42.13.1 Method overview . . . 949
42.13.2 TStringWriter.Create . . . 949
42.13.3 TStringWriter.Destroy . . . 949
42.13.4 TStringWriter.Close . . . 949
42.13.5 TStringWriter.Flush . . . 949
42.13.6 TStringWriter.Write . . . 949
42.13.7 TStringWriter.WriteLine . . . 950
42.13.8 TStringWriter.ToString . . . 950
42.14TTextReader . . . 950
42.14.1 Description . . . 950
42.14.2 Method overview . . . 950
42.14.3 Property overview . . . 951
42.14.4 TTextReader.Create . . . 951
42.14.5 TTextReader.Reset . . . 951
42.14.6 TTextReader.Close . . . 951
42.14.7 TTextReader.ReadLine . . . 951
42.14.8 TTextReader.Eof . . . 952
42.15TTextWriter . . . 952
42.15.1 Method overview . . . 952
42.15.2 TTextWriter.Close . . . 952
42.15.3 TTextWriter.Flush . . . 952

42.15.4 TTextWriter.Write . . . 952
42.15.5 TTextWriter.WriteLine . . . 953
42.16TWindowedStream . . . 953
42.16.1 Description . . . 953
42.16.2 Method overview . . . 953
42.16.3 TWindowedStream.Create . . . 954
42.16.4 TWindowedStream.Destroy . . . 954
42.16.5 TWindowedStream.Read . . . 954
42.16.6 TWindowedStream.Write . . . 954
42.16.7 TWindowedStream.Seek . . . 955

43 Reference for unit 'StreamIO' 956
43.1 Used units . . . 956
43.2 Overview . . . 956
43.3 Procedures and functions . . . 956
43.3.1 AssignStream . . . 956
43.3.2 GetStream . . . 957

44 Reference for unit 'syncobjs' 958
44.1 Used units . . . 958
44.2 Overview . . . 958
44.3 Constants, types and variables . . . 958
44.3.1 Constants . . . 958
44.3.2 Types . . . 958
44.4 ELockException . . . 959
44.4.1 Description . . . 959
44.5 ELockRecursionException . . . 959
44.5.1 Description . . . 959
44.6 ESyncObjectException . . . 959
44.6.1 Description . . . 959
44.7 TCriticalSection . . . 959
44.7.1 Description . . . 959
44.7.2 Method overview . . . 960
44.7.3 TCriticalSection.Acquire . . . 960
44.7.4 TCriticalSection.Release . . . 960
44.7.5 TCriticalSection.Enter . . . 961
44.7.6 TCriticalSection.TryEnter . . . 961
44.7.7 TCriticalSection.Leave . . . 961
44.7.8 TCriticalSection.Create . . . 961
44.7.9 TCriticalSection.Destroy . . . 962
44.8 TEventObject . . . 962

44.8.1 Description . . . 962
44.8.2 Method overview . . . 962
44.8.3 Property overview . . . 962
44.8.4 TEventObject.Create . . . 962
44.8.5 TEventObject.destroy . . . 963
44.8.6 TEventObject.ResetEvent . . . 963
44.8.7 TEventObject.SetEvent . . . 963
44.8.8 TEventObject.WaitFor . . . 963
44.8.9 TEventObject.ManualReset . . . 964
44.9 THandleObject . . . 964
44.9.1 Description . . . 964
44.9.2 Method overview . . . 964
44.9.3 Property overview . . . 964
44.9.4 THandleObject.destroy . . . 965
44.9.5 THandleObject.Handle . . . 965
44.9.6 THandleObject.LastError . . . 965
44.10 TSimpleEvent . . . 965
44.10.1 Description . . . 965
44.10.2 Method overview . . . 965
44.10.3 TSimpleEvent.Create . . . 966
44.11 TSynchroObject . . . 966
44.11.1 Description . . . 966
44.11.2 Method overview . . . 966
44.11.3 TSynchroObject.Acquire . . . 966
44.11.4 TSynchroObject.Release . . . 966

45 Reference for unit 'URIParser' 967
45.1 Used units . . . 967
45.2 Overview . . . 967
45.3 Constants, types and variables . . . 967
45.3.1 Types . . . 967
45.4 Procedures and functions . . . 967
45.4.1 EncodeURI . . . 967
45.4.2 FilenameToURI . . . 968
45.4.3 IsAbsoluteURI . . . 968
45.4.4 ParseURI . . . 968
45.4.5 ResolveRelativeURI . . . 968
45.4.6 URIToFilename . . . 969
45.5 TURI . . . 969

46 Reference for unit 'Zipper' 970

46.1 Used units . . . 970
46.2 Overview . . . 970
46.3 Constants, types and variables . . . 970
46.3.1 Constants . . . 970
46.3.2 Types . . . 974
46.4 Central_File_Header_Type . . . 975
46.5 CodeRec . . . 976
46.6 End_of_Central_Dir_Type . . . 976
46.7 Extensible_Data_Field_Header_Type . . . 976
46.8 Local_File_Header_Type . . . 977
46.9 Zip64_End_of_Central_Dir_Locator_type . . . 977
46.10Zip64_End_of_Central_Dir_type . . . 977
46.11Zip64_Extended_Info_Field_Type . . . 978
46.12EZipError . . . 978
46.12.1 Description . . . 978
46.13TCompressor . . . 978
46.13.1 Description . . . 978
46.13.2 Method overview . . . 978
46.13.3 Property overview . . . 978
46.13.4 TCompressor.Create . . . 979
46.13.5 TCompressor.Compress . . . 979
46.13.6 TCompressor.ZipID . . . 979
46.13.7 TCompressor.ZipVersionReqd . . . 979
46.13.8 TCompressor.ZipBitFlag . . . 979
46.13.9 TCompressor.Terminate . . . 979
46.13.10TCompressor.BufferSize . . . 980
46.13.11TCompressor.OnPercent . . . 980
46.13.12TCompressor.OnProgress . . . 980
46.13.13TCompressor.Crc32Val . . . 980
46.13.14TCompressor.Terminated . . . 980
46.14TDeCompressor . . . 981
46.14.1 Description . . . 981
46.14.2 Method overview . . . 981
46.14.3 Property overview . . . 981
46.14.4 TDeCompressor.Create . . . 981
46.14.5 TDeCompressor.DeCompress . . . 981
46.14.6 TDeCompressor.Terminate . . . 981
46.14.7 TDeCompressor.ZipID . . . 982
46.14.8 TDeCompressor.BufferSize . . . 982
46.14.9 TDeCompressor.OnPercent . . . 982

46.14.10TDeCompressor.OnProgress . . . 982
46.14.11TDeCompressor.OnProgressEx . . . 982
46.14.12TDeCompressor.Crc32Val . . . 982
46.14.13TDeCompressor.Terminated . . . 983
46.15TDeflater . . . 983
46.15.1 Description . . . 983
46.15.2 Method overview . . . 983
46.15.3 Property overview . . . 983
46.15.4 TDeflater.Create . . . 983
46.15.5 TDeflater.Compress . . . 983
46.15.6 TDeflater.ZipID . . . 984
46.15.7 TDeflater.ZipVersionReqd . . . 984
46.15.8 TDeflater.ZipBitFlag . . . 984
46.15.9 TDeflater.CompressionLevel . . . 984
46.16TFullZipFileEntries . . . 985
46.16.1 Description . . . 985
46.16.2 Property overview . . . 985
46.16.3 TFullZipFileEntries.FullEntries . . . 985
46.17TFullZipFileEntry . . . 985
46.17.1 Description . . . 985
46.17.2 Property overview . . . 985
46.17.3 TFullZipFileEntry.BitFlags . . . 986
46.17.4 TFullZipFileEntry.CompressMethod . . . 986
46.17.5 TFullZipFileEntry.CompressedSize . . . 986
46.17.6 TFullZipFileEntry.CRC32 . . . 986
46.18TInflater . . . 986
46.18.1 Description . . . 986
46.18.2 Method overview . . . 987
46.18.3 TInflater.Create . . . 987
46.18.4 TInflater.DeCompress . . . 987
46.18.5 TInflater.ZipID . . . 987
46.19TShrinker . . . 987
46.19.1 Description . . . 987
46.19.2 Method overview . . . 987
46.19.3 TShrinker.Create . . . 988
46.19.4 TShrinker.Destroy . . . 988
46.19.5 TShrinker.Compress . . . 988
46.19.6 TShrinker.ZipID . . . 988
46.19.7 TShrinker.ZipVersionReqd . . . 988
46.19.8 TShrinker.ZipBitFlag . . . 988

46.20TUnZipper . . . 989
46.20.1 Description . . . 989
46.20.2 Method overview . . . 989
46.20.3 Property overview . . . 989
46.20.4 TUnZipper.Create . . . 990
46.20.5 TUnZipper.Destroy . . . 990
46.20.6 TUnZipper.UnZipAllFiles . . . 990
46.20.7 TUnZipper.UnZipFile . . . 990
46.20.8 TUnZipper.UnZipFiles . . . 991
46.20.9 TUnZipper.Unzip . . . 991
46.20.10TUnZipper.Clear . . . 991
46.20.11TUnZipper.Examine . . . 992
46.20.12TUnZipper.Terminate . . . 992
46.20.13TUnZipper.BufferSize . . . 992
46.20.14TUnZipper.OnOpenInputStream . . . 992
46.20.15TUnZipper.OnCloseInputStream . . . 992
46.20.16TUnZipper.OnCreateStream . . . 993
46.20.17TUnZipper.OnDoneStream . . . 993
46.20.18TUnZipper.OnPercent . . . 993
46.20.19TUnZipper.OnProgress . . . 993
46.20.20TUnZipper.OnProgressEx . . . 993
46.20.21TUnZipper.OnStartFile . . . 994
46.20.22TUnZipper.OnEndFile . . . 994
46.20.23TUnZipper.FileName . . . 994
46.20.24TUnZipper.OutputPath . . . 994
46.20.25TUnZipper.FileComment . . . 994
46.20.26TUnZipper.Files . . . 995
46.20.27TUnZipper.Entries . . . 995
46.20.28TUnZipper.UseUTF8 . . . 995
46.20.29TUnZipper.Flat . . . 996
46.20.30TUnZipper.Terminated . . . 996
46.21TZipFileEntries . . . 996
46.21.1 Description . . . 996
46.21.2 Method overview . . . 996
46.21.3 Property overview . . . 996
46.21.4 TZipFileEntries.AddFileEntry . . . 996
46.21.5 TZipFileEntries.AddFileEntries . . . 997
46.21.6 TZipFileEntries.Entries . . . 997
46.22TZipFileEntry . . . 997
46.22.1 Description . . . 997

46.22.2 Method overview . . . 998
46.22.3 Property overview . . . 998
46.22.4 TZipFileEntry.Create . . . 998
46.22.5 TZipFileEntry.IsDirectory . . . 999
46.22.6 TZipFileEntry.IsLink . . . 999
46.22.7 TZipFileEntry.Assign . . . 999
46.22.8 TZipFileEntry.Stream . . . 999
46.22.9 TZipFileEntry.ArchiveFileName . . . 1000
46.22.10 TZipFileEntry.UTF8ArchiveFileName . . . 1000
46.22.11 TZipFileEntry.DiskFileName . . . 1000
46.22.12 TZipFileEntry.UTF8DiskFileName . . . 1000
46.22.13 TZipFileEntry.Size . . . 1000
46.22.14 TZipFileEntry.DateTime . . . 1001
46.22.15 TZipFileEntry.OS . . . 1001
46.22.16 TZipFileEntry.Attributes . . . 1001
46.22.17 TZipFileEntry.CompressionLevel . . . 1001
46.23 TZipper . . . 1001
46.23.1 Description . . . 1001
46.23.2 Method overview . . . 1002
46.23.3 Property overview . . . 1002
46.23.4 TZipper.Create . . . 1002
46.23.5 TZipper.Destroy . . . 1002
46.23.6 TZipper.ZipAllFiles . . . 1003
46.23.7 TZipper.SaveToFile . . . 1003
46.23.8 TZipper.SaveToStream . . . 1003
46.23.9 TZipper.ZipFile . . . 1004
46.23.10 TZipper.ZipFiles . . . 1004
46.23.11 TZipper.Zip . . . 1004
46.23.12 TZipper.Clear . . . 1005
46.23.13 TZipper.Terminate . . . 1005
46.23.14 TZipper.BufferSize . . . 1005
46.23.15 TZipper.OnPercent . . . 1005
46.23.16 TZipper.OnProgress . . . 1006
46.23.17 TZipper.OnStartFile . . . 1006
46.23.18 TZipper.OnEndFile . . . 1006
46.23.19 TZipper.FileName . . . 1006
46.23.20 TZipper.FileComment . . . 1006
46.23.21 TZipper.Files . . . 1007
46.23.22 TZipper.InMemSize . . . 1007
46.23.23 TZipper.Entries . . . 1007

46.23.24 TZipper.Terminated . . . 1007
46.23.25 TZipper.UseLanguageEncoding . . . 1008

47 Reference for unit 'ZStream' 1009
47.1 Used units . . . 1009
47.2 Overview . . . 1009
47.3 Constants, types and variables . . . 1009
47.3.1 Types . . . 1009
47.4 Ecompressionerror . . . 1010
47.4.1 Description . . . 1010
47.5 Edecompressionerror . . . 1010
47.5.1 Description . . . 1010
47.6 Egzfileerror . . . 1010
47.6.1 Description . . . 1010
47.7 Ezliberror . . . 1010
47.7.1 Description . . . 1010
47.8 Tcompressionstream . . . 1010
47.8.1 Description . . . 1010
47.8.2 Method overview . . . 1011
47.8.3 Property overview . . . 1011
47.8.4 Tcompressionstream.create . . . 1011
47.8.5 Tcompressionstream.destroy . . . 1011
47.8.6 Tcompressionstream.write . . . 1011
47.8.7 Tcompressionstream.flush . . . 1012
47.8.8 Tcompressionstream.get_compressionrate . . . 1012
47.8.9 Tcompressionstream.OnProgress . . . 1012
47.9 Tcustomzlibstream . . . 1012
47.9.1 Description . . . 1012
47.9.2 Method overview . . . 1013
47.9.3 Tcustomzlibstream.create . . . 1013
47.9.4 Tcustomzlibstream.destroy . . . 1013
47.10 Tdecompressionstream . . . 1013
47.10.1 Description . . . 1013
47.10.2 Method overview . . . 1013
47.10.3 Property overview . . . 1014
47.10.4 Tdecompressionstream.create . . . 1014
47.10.5 Tdecompressionstream.destroy . . . 1014
47.10.6 Tdecompressionstream.read . . . 1014
47.10.7 Tdecompressionstream.Seek . . . 1015
47.10.8 Tdecompressionstream.get_compressionrate . . . 1015

47.10.9 Tdecompressionstream.OnProgress 1015
47.11TGZFileStream 1016
47.11.1 Description 1016
47.11.2 Method overview 1016
47.11.3 TGZFileStream.create 1016
47.11.4 TGZFileStream.read 1016
47.11.5 TGZFileStream.write 1017
47.11.6 TGZFileStream.seek 1017
47.11.7 TGZFileStream.destroy 1017
47.12TGZipCompressionStream 1017
47.12.1 Method overview 1017
47.12.2 TGZipCompressionStream.Create 1017
47.12.3 TGZipCompressionStream.Destroy 1018
47.12.4 TGZipCompressionStream.Write 1018
47.13TGZipDecompressionStream 1018
47.13.1 Method overview 1018
47.13.2 TGZipDecompressionStream.Create 1018
47.13.3 TGZipDecompressionStream.Destroy 1018
47.13.4 TGZipDecompressionStream.Read 1018
47.13.5 TGZipDecompressionStream.Seek 1018

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 'AdvancedSingleInstance'

1.1 Used units

Table 1.1: Used units by unit 'AdvancedSingleInstance'

Name	Page
AdvancedIPC	**??**
Classes	**??**
singleinstance	838
System	**??**
sysutils	**??**

1.2 Overview

The `AdvancedSingleInstance` unit contains an implementation of the TBaseSingleInstance (839) class, based on functionality found in the AdvcancedIPC (**??**) unit. The class is called TAdvancedSingleInstance (96)

1.3 Constants, types and variables

1.3.1 Types

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

`TSingleInstanceReceivedCustomMessage` is the signature of the TAdvancedSingleInstance.OnServerReceivedCustomRequest (99) event. It carries the information received when the server receives a custom message.

1.4 TAdvancedSingleInstance

1.4.1 Description

`TAdvancedSingleInstance` is a TBaseSingleInstance (95) descendent. It implements the required functionality using the IPC mechanism of the advancedipc (95) unit: it uses files to communicate. This ensures the mechanism works on all platforms.

See also: TBaseSingleInstance (95), advancedipc (95)

1.4.2 Method overview

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

1.4.3 Property overview

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

1.4.4 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 (95) property which identifies the application.

See also: ID (95)

1.4.5 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 (95) call. It initiates the IPC mechanism.

See also: TBaseSingleInstance.Start (95), TAdvancedSingleInstance.Stop (97)

1.4.6 TAdvancedSingleInstance.Stop

Synopsis: End IPC communication.

Declaration: `procedure Stop; Override`

Visibility: public

Description: `Start` implements the abstract TBaseSingleInstance.Stop (95) call. It terminates the IPC mechanism.

See also: TBaseSingleInstance.Stop (95), TAdvancedSingleInstance.Start (97)

1.4.7 TAdvancedSingleInstance.ServerCheckMessages

Synopsis: Check for incoming messages for the server.

Declaration: `procedure ServerCheckMessages; Override`

Visibility: public

Description: `ServerCheckMessages` implements TAdvancedSingleInstance.ServerCheckMessages (97) and checks if there are any messages in the message queue using the API of advancedIPC (95). If there are messages, it fires the OnServerReceivedParams (838) event.

See also: TAdvancedSingleInstance.ServerCheckMessages (97), TAdvancedSingleInstance.Start (97), TAdvancedSingleInstance.Stop (97), OnServerReceivedParams (838)

1.4.8 TAdvancedSingleInstance.ClientPostParams

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

Declaration: `procedure ClientPostParams; Override`

Visibility: public

Description: `ClientPostParams` implements TAdvancedSingleInstance.ClientPostParams (97). It sends the command-line parameters using ClientPostCustomRequest (98).

Errors: If this application instance is not a client instance, then a ESingleInstance (95) exception is raised.

See also: TAdvancedSingleInstance.ClientPostParams (97), ClientPostCustomRequest (98)

1.4.9 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 (95) exception is raised.

See also: TAdvancedSingleInstance.ClientSendCustomRequest (98), TAdvancedSingleInstance.ServerPostCustomResponse (98)

1.4.10 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 (95) exception is raised.

See also: TAdvancedSingleInstance.ClientPostCustomRequest (98), TAdvancedSingleInstance.ServerPostCustomResponse (98)

1.4.11 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 (99)

Errors: If this application instance is not a server instance, then a ESingleInstance (95) exception is raised.

See also: TAdvancedSingleInstance.ClientPostCustomRequest (98), TAdvancedSingleInstance.ClientSendCustomRequest (98), TAdvancedSingleInstance.ClientPeekCustomResponse (99)

1.4.12 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 (95) 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 (95) exception is raised. The `aStream` must be non-nil, or an access violation will occur.

See also: TimeoutMessages (95), TAdvancedSingleInstance.ServerPostCustomResponse (98)

1.4.13 TAdvancedSingleInstance.ID

Synopsis: The ID of this instance.

Declaration:
```
Property ID : string
```

Visibility: public

Access: Read,Write

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

See also: TBaseSingleInstance.ID (95)

1.4.14 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 (97) is called.

See also: Start (97)

1.4.15 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 (95)

Chapter 2

Reference for unit 'ascii85'

2.1 Used units

Table 2.1: Used units by unit 'ascii85'

Name	Page
Classes	??
System	??
sysutils	??

2.2 Overview

The ascii85 provides an ASCII 85 or base 85 decoding algorithm. It is class and stream based: the TASCII85DecoderStream (101) 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 (669) or base64 (129) units.

2.3 Constants, types and variables

2.3.1 Types

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

Table 2.2: 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.

2.4 TASCII85DecoderStream

2.4.1 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.

2.4.2 Method overview

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

2.4.3 Property overview

Page	Properties	Access	Description
103	BExpectBoundary	rw	Expect c̃haracter.

2.4.4 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 (102)

2.4.5 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 (102)

2.4.6 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 (102), TASCII85DecoderStream.Read (103), TASCII85DecoderStream.Destroy (102)

2.4.7 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 (102), TASCII85DecoderStream.BExpectBoundary (103)

2.4.8 TASCII85DecoderStream.Destroy

Synopsis: Clean up instance.

Declaration: `destructor Destroy;  Override`

Visibility: public

Description: `Destroy` closes the input stream using Close (102) and cleans up the `TASCII85DecoderStream` instance from memory.

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

See also: TASCII85DecoderStream.Close (102)

2.4.9 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.

2.4.10 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 (103)

2.4.11 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 (102)

2.5 TASCII85EncoderStream

2.5.1 Description

`TASCII85EncoderStream` is the counterpart to the TASCII85DecoderStream (101) decoder stream: what `TASCII85EncoderStream` encodes, can be decoded by TASCII85DecoderStream (101).

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 (104), TASCII85DecoderStream (101)

2.5.2 Method overview

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

2.5.3 Property overview

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

2.5.4 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 (103), Width (105), Boundary (105)

2.5.5 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 (104), TASCII85EncoderStream.Boundary (105)

2.5.6 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 (104)

2.5.7 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 (105), Create (104)

2.5.8 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 (105), Create (104)

2.6 TASCII85RingBuffer

2.6.1 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 (101) decoder stream will create an instance of this class automatically.

See also: TASCII85DecoderStream (101)

2.6.2 Method overview

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

2.6.3 Property overview

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

2.6.4 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 (106), TASCII85RingBuffer.Read (106), TASCII85RingBuffer.Size (106)

2.6.5 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 (106), TASCII85RingBuffer.Write (106), TASCII85RingBuffer.Size (106)

2.6.6 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 (106), TASCII85RingBuffer.Read (106), TASCII85RingBuffer.Size (106)

2.6.7 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 (106)

Chapter 3

Reference for unit 'AVL_Tree'

3.1 Used units

Table 3.1: Used units by unit 'AVL_Tree'

Name	Page
Classes	??
System	??
sysutils	??

3.2 Overview

The `avl_tree` unit implements a general-purpose AVL (balanced) tree class: the TAVLTree (108) class and it's associated data node class TAVLTreeNode (122).

3.3 Constants, types and variables

3.3.1 Types

```
PAVLTreeNode = ^TAVLTreeNode
```

Pointer to TAVLTreeNode.

```
TAVLTreeClass = Class of TAVLTree
```

`TAVLTreeClass` is the class of TAVLTree (108).

```
TAVLTreeNodeClass = Class of TAVLTreeNode
```

`TAVLTreeNodeClass` is the class of TAVLTreeNode (122). It is the type of the TAVLTree.NodeClass (121) 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 (121) 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 (121): it also passes the tree to the sort mechanism.

3.3.2 Variables

NodeMemManager : TAVLTreeNodeMemManager

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

3.4 TAVLTree

3.4.1 Description

TAVLTree maintains a balanced AVL tree. The tree consists of TAVLTreeNode (122) 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 (121) property.

See also: TAVLTreeNode (122)

3.4.2 Method overview

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

3.4.3 Property overview

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

3.4.4 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 (108). An alternate OnCompare (121) can be provided: the default `OnCompare` method compares the 2 data pointers of a node.

See also: OnCompare (121)

3.4.5 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 (108) compare function instead of a regular `TListSortCompare` compare function. The compare function can still be set in the TAVLTree.OnObjectCompare (121) property.

See also: TAVLTree.OnObjectCompare (121)

3.4.6 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 (113), TAVLTree.Create (110)

3.4.7 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 (125) is used to manage the nodes of all `TAVLTree` instances.

See also: TBaseAVLTreeNodeManager (127), TAVLTreeNodeMemManager (125)

3.4.8 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 (121) class is used to create a new node.

See also: TAVLTree.NodeClass (121), TAVLTree.Add (111), TAVLTree.DisposeNode (111)

3.4.9 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 (111)

3.4.10 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 (112), TAVLTree.Remove (112)

3.4.11 TAVLTree.AddAscendingSequence

Synopsis:

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

Visibility: public

Description: `AddAscendingSequence` is an optimized version of Add (111) 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 (111)

3.4.12 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 (125) 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 (112), TAVLTree.RemovePointer (113), TAVLTree.Clear (113)

3.4.13 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 (115) and, if found, deletes it from the tree. Only the first occurrence of `Data` will be removed.

See also: TAVLTree.Delete (112), TAVLTree.RemovePointer (113), TAVLTree.Clear (113), TAVLTree.Find (115)

3.4.14 TAVLTree.RemovePointer

Synopsis: Remove a pointer item from the list.

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

Visibility: public

Description: `Remove` uses FindPointer (117) 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 (112), TAVLTree.Delete (112), TAVLTree.Clear (113)

3.4.15 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 (113)

3.4.16 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 (113)

3.4.17 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 (114) instead.

See also: TAVLTree.FreeAndClear (114), TAVLTree.Delete (112)

3.4.18 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 (113), TAVLTree.Delete (112), TAVLTree.FreeAndDelete (114)

3.4.19 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 (113), TAVLTree.Delete (112), TAVLTree.FreeAndClear (114)

3.4.20 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 (114), comparing keys only). If another type of object is passed, the inherited `IsEqual` is called.

Errors: None.

See also: TAvlTree.IsEqual (114)

3.4.21 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 (114)

3.4.22 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 (114)

3.4.23 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 (121) or TAVLTree.OnObjectCompare (121) 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 (121), TAVLtree.OnCompare (121)

3.4.24 TAVLTree.Find

Synopsis: Find a data item in the tree.

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

Visibility: public

Description: `Find` uses the default OnCompare (121) 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 (117).

See also: OnCompare (121), FindKey (115)

3.4.25 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 (121), Find (115)

3.4.26 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 (115)

3.4.27 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 (116), TAVLTree.MoveDataRightMost (113)

3.4.28 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 (116), TAVLTree.MoveDataLeftMost (113)

3.4.29 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 (117)

3.4.30 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 (117)

3.4.31 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 (117), FindLowest (117), Find (115), FindKey (115)

3.4.32 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 (115), where a custom compare function can be used.

The default OnCompare (121) compares the data pointers, so the default `Find` will return the same node as `FindPointer`

See also: TAVLTree.Find (115), TAVLTree.FindKey (115)

3.4.33 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 (118), TAVLTree.FindLeftMostKey (118), TAVLTree.FindRightMostKey (118)

3.4.34 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 (118), TAVLTree.FindLeftMostKey (118), TAVLTree.FindRightMostKey (118)

3.4.35 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 (118), TAVLTree.FindRightMost (118), TAVLTree.FindRightMostKey (118)

3.4.36 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 (118), TAVLTree.FindRightMost (118), TAVLTree.FindLeftMostKey (118)

3.4.37 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 (118), TAVLTree.FindLeftMostKey (118), TAVLTree.FindRightMostSameKey (119)

3.4.38 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 (118), TAVLTree.FindRightMostKey (118), TAVLTree.FindLeftMostSameKey (119)

3.4.39 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 (123).

See also: TAVLTreeNodeEnumerator (123)

3.4.40 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 (119)

3.4.41 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 (120)

3.4.42 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 (120)

3.4.43 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 (120) 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 (120)

3.4.44 TAVLTree.ReportAsString

Synopsis: Return the tree report as a string.

Declaration: `function ReportAsString : string`

Visibility: public

Description: `ReportAsString` calls WriteReportToStream (120) and returns the stream data as a string.

See also: TAVLTree.WriteReportToStream (120)

3.4.45 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 (110)

3.4.46 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 (121) is not set.

See also: TAVLTree.OnCompare (121)

3.4.47 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 (111) will use this class when creating a new node. This can be set to a descendent class of TAVLTreeNode (122), but not if there are already nodes in the tree.

See also: TAVLTreeNode (122), TAVLTree.NewNode (111)

3.4.48 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 (111), Delete (112), Remove (112), RemovePointer (113), or Clear (113) methods to manipulate the items in the tree.

See also: TAVLTree.Add (111), TAVLTree.Delete (112), TAVLTree.Remove (112), TAVLTree.RemovePointer (113), TAVLTree.Clear (113)

3.4.49 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.

3.5 TAVLTreeNode

3.5.1 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 (108), TAVLTreeNode.Data (**??**)

3.5.2 Method overview

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

3.5.3 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 (122)

3.5.4 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 (122)

3.5.5 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.

3.5.6 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 (**??**)

3.5.7 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.

3.5.8 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.

3.6 TAVLTreeNodeEnumerator

3.6.1 Description

`TAVLTreeNodeEnumerator` is a class which implements the enumerator interface for the TAVL-Tree (108). It enumerates all the nodes in the tree.

See also: TAVLTree (108)

3.6.2 Method overview

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

3.6.3 Property overview

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

3.6.4 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.

3.6.5 TAVLTreeNodeEnumerator.GetEnumerator

Synopsis: Returns the enumerator.

Declaration: `function GetEnumerator : TAVLTreeNodeEnumerator`

Visibility: public

Description: `GetEnumerator` returns `Self`..

3.6.6 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 (116).

See also: TAVLTree.FindSuccessor (116)

3.6.7 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 (124)

3.6.8 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 (124)

3.7 TAVLTreeNodeMemManager

3.7.1 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 (122), TAVLTree (108)

3.7.2 Method overview

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

3.7.3 Property overview

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

3.7.4 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 (126)

3.7.5 TAVLTreeNodeMemManager.NewNode

Synopsis: Create a new `TAVLTreeNode` instance.

Declaration: `function NewNode : TAVLTreeNode;  Override`

Visibility: public

Description: `NewNode` returns a new TAVLTreeNode (122) 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 (125)

3.7.6 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 (127), TAVLTreeNodeMemManager.MaximumFreeNodeRatio (127)

3.7.7 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 (126)

3.7.8 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 (126)

3.7.9 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 (127)

3.7.10 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 (127), TAVLTreeNodeMemManager.MinimumFreeNode (127)

3.7.11 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 (127), TAVLTreeNodeMemManager.MaximumFreeNodeRatio (127)

3.8 TBaseAVLTreeNodeManager

3.8.1 Description

`TBaseAVLTreeNodeManager` is an abstract class from which a descendent can be created that manages creating and disposing of tree nodes (instances of TAVLTreeNode (122)) for a TAVLTree (108) 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 (125).

The TAVLTree.SetNodeManager (110) method can be used to set the node manager that a `TAVLTree` instance should use.

See also: TAVLTreeNodeMemManager (125), TAVLTree.SetNodeManager (110), TAVLTreeNode (122)

3.8.2 Method overview

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

3.8.3 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 (108) when it no longer needs a TAVLTreeNode (122) instance. The manager may decide to re-use the instance for later use instead of destroying it.

See also: TBaseAVLTreeNodeManager.NewNode (128), TAVLTree.Delete (112), TAVLTreeNode (122)

3.8.4 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 (108) when it needs a new node in TAVLTree.Add (111). It must be implemented by descendants to return a new TAVLTreeNode (122) instance.

See also: TBaseAVLTreeNodeManager.DisposeNode (128), TAVLTree.Add (111), TAVLTreeNode (122)

Chapter 4

Reference for unit 'base64'

4.1 Used units

Table 4.1: Used units by unit 'base64'

Name	Page
Classes	??
System	??
sysutils	??

4.2 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 (133) for encoding, and TBase64DecodingStream (130) 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...

4.3 Constants, types and variables

4.3.1 Types

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

Table 4.2: Enumeration values for type TBase64DecodingMode

Value	Explanation
bdmMIME	MIME encoding.
bdmStrict	Strict encoding.

TBase64DecodingMode determines the decoding algorithm used by TBase64DecodingStream (130). 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 (130) 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.

4.4 Procedures and functions

4.4.1 DecodeStringBase64

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

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

Visibility: default

Description: DecodeStringBase64 decodes the string s (containing Base 64 encoded data) returns the decoded data as a string. It uses a TBase64DecodingStream (130) to do this. The Strict parameter is passed on to the constructor as bdmStrict or bdmMIME

See also: DecodeStringBase64 (130), TBase64DecodingStream (130)

4.4.2 EncodeStringBase64

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

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

Visibility: default

Description: EncodeStringBase64 encodes the string s using Base 64 encoding and returns the result. It uses a TBase64EncodingStream (133) to do this.

See also: DecodeStringBase64 (130), TBase64EncodingStream (133)

4.5 EBase64DecodingException

4.5.1 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.

4.6 TBase64DecodingStream

4.6.1 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 (133)

4.6.2 Method overview

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

4.6.3 Property overview

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

4.6.4 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 (133), TBase64DecodingMode (129)

4.6.5 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 (132), TBase64DecodingStream.Read (131)

4.6.6 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 (132), TStream.Read (**??**)

4.6.7 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 (131), TBase64EncodingStream.Seek (134), TStream.Seek (**??**)

4.6.8 TBase64DecodingStream.EOF

Synopsis:

Declaration: `Property EOF : Boolean`

Visibility: public

Access: Read

Description:

4.6.9 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 (130)

4.7 TBase64EncodingStream

4.7.1 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 (133) 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 (130)

4.7.2 Method overview

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

4.7.3 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.

4.7.4 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 (133)

4.7.5 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 (134), TStream.Write (**??**)

4.7.6 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 (134), TStream.Seek (**??**)

Chapter 5

Reference for unit 'BlowFish'

5.1 Used units

Table 5.1: Used units by unit 'BlowFish'

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

5.2 Overview

The BlowFish implements a class TBlowFish (136) to handle Blowfish encryption/decryption of memory buffers, and 2 TStream (**??**) descendents TBlowFishDeCryptStream (137) which decrypts any data that is read from it on the fly, as well as TBlowFishEnCryptStream (138) which encrypts the data that is written to it on the fly.

5.3 Constants, types and variables

5.3.1 Constants

```
BFRounds = 16
```

Number of rounds in Blowfish encryption.

5.3.2 Types

```
PBlowFishKey = ^TBlowFishKey
```

`PBlowFishKey` is a simple pointer to a TBlowFishKey (136) array.

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

TBFBlock is the basic data structure used by the encrypting/decrypting routines in TBlowFish (136), TBlowFishDeCryptStream (137) and TBlowFishEnCryptStream (138). 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 (136), TBlowFishDeCryptStream (137) and TBlowFishEnCryptStream (138) classes. It should be filled with the encryption key and passed to the constructor of one of these classes.

5.4 EBlowFishError

5.4.1 Description

EBlowFishError is used by the TBlowFishStream (140), TBlowFishEncryptStream (138) and TBlowFishDecryptStream (137) classes to report errors.

See also: TBlowFishStream (140), TBlowFishEncryptStream (138), TBlowFishDecryptStream (137)

5.5 TBlowFish

5.5.1 Description

TBlowFish is a simple class that can be used to encrypt/decrypt a single TBFBlock (136) data block with the Encrypt (137) and Decrypt (137) calls. It is used internally by the TBlowFishEnCryptStream (138) and TBlowFishDeCryptStream (137) classes to encrypt or decrypt the actual data.

See also: TBlowFishEnCryptStream (138), TBlowFishDeCryptStream (137)

5.5.2 Method overview

Page	Method	Description
136	Create	Create a new instance of the TBlowFish class.
137	Decrypt	Decrypt a block.
137	Encrypt	Encrypt a block.

5.5.3 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 (137) and Decrypt (137).

See also: Encrypt (137), Decrypt (137)

5.5.4 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 (136) specified when the `TBlowFish` instance was created.

See also: TBlowFishKey (136), Decrypt (137), Create (136)

5.5.5 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 (136) specified when the `TBlowFish` instance was created. The data must have been encrypted with the same key and the Encrypt (137) call.

See also: TBlowFishKey (136), Encrypt (137), Create (136)

5.6 TBlowFishDeCryptStream

5.6.1 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 (138) stream must be used.

See also: Create (140), TBlowFishEncryptStream (138)

5.6.2 Method overview

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

5.6.3 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 (138), TBlowFishDeCryptStream.Seek (138), TBlowFishEnCryptStream.Write (139)

5.6.4 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 (140), TBlowFishEncryptStream (138)

5.6.5 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 (136) exception will be raised.

See also: Read (138), EBlowFishError (136)

5.7 TBlowFishEncryptStream

5.7.1 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 (137) stream must be used.

See also: Create (140), TBlowFishDecryptStream (137)

5.7.2 Method overview

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

5.7.3 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 (140), Create (140)

5.7.4 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 (140) mechanism can be used to write the remaining bytes.

See also: TBlowFishEncryptStream.Flush (140)

5.7.5 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 (136) exception.

See also: TBlowFishEncryptStream.Write (139)

5.7.6 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 (139), TBFBlock (136)

5.8 TBlowFishStream

5.8.1 Description

`TBlowFishStream` is an abstract class which is used as a parent class for TBlowFishEncryptStream (138) and TBlowFishDecryptStream (137). It simply provides a constructor and storage for a TBlowFish (136) 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 (138), TBlowFishDecryptStream (137), TBlowFish (136)

5.8.2 Method overview

Page	Method	Description
140	Create	Create a new instance of the `TBlowFishStream` class.
141	Destroy	Destroy the `TBlowFishStream` instance.

5.8.3 Property overview

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

5.8.4 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 (136) 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 (138), TBlowFishDecryptStream (137), TBlowFish (136)

5.8.5 TBlowFishStream.Destroy

Synopsis: Destroy the `TBlowFishStream` instance.

Declaration: `destructor Destroy;  Override`

Visibility: public

Description: `Destroy` cleans up the internal TBlowFish (136) instance.

See also: Create (140), TBlowFish (136)

5.8.6 TBlowFishStream.BlowFish

Synopsis: Blowfish instance used when encrypting/decrypting.

Declaration: `Property BlowFish : TBlowFish`

Visibility: public

Access: Read

Description: `BlowFish` is the TBlowFish (136) 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 (138) and TBlowFishDecryptStream (137).

See also: TBlowFishEncryptStream (138), TBlowFishDecryptStream (137), TBlowFish (136)

Chapter 6

Reference for unit 'BufDataset'

6.1 Used units

Table 6.1: Used units by unit 'BufDataset'

Name	Page
bufdataset_parser	??
Classes	??
DB	351
System	??
sysutils	??

6.2 Overview

Implements a database-aware buffered dataset.

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

6.3 Constants, types and variables

6.3.1 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 6.2: 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 6.3:

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 6.4: 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).

6.4 Procedures and functions

6.4.1 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 (143), TDataPacketFormat (143), TXMLDatapacketReader (351)

6.5 TApplyRecUpdateResult

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

```
  Async : Boolean;
end
```

6.6 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.

6.7 TBufBlobField

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

Implements a Blob field for TBufDataset.

6.8 TBufBookmark

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

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

6.9 TBufRecLinkItem

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

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

6.10 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.

6.11 TRecUpdateBuffer

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

Implements a buffer used for record updates in TBufDataset.

6.12 TArrayBufIndex

6.12.1 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 (158), TRecordBuffer (142), TBufBookmark (146)

6.12.2 Method overview

Page	Method	Description
151	AddRecord	
151	BeginUpdate	
150	CanScrollForward	
148	Create	
150	DoScrollForward	
151	EndUpdate	
148	GetCurrent	
150	GotoBookmark	
150	InitialiseIndex	
150	InitialiseSpareRecord	
151	InsertRecordBeforeCurrentRecord	
151	ReleaseSpareRecord	
151	RemoveRecordFromIndex	
149	RestoreCurrentRecord	
148	ScrollBackward	
149	ScrollFirst	
148	ScrollForward	
149	ScrollLast	
149	SetToFirstRecord	
149	SetToLastRecord	
150	StoreCurrentRecIntoBookmark	
149	StoreCurrentRecord	
150	StoreSpareRecIntoBookmark	

6.12.3 TArrayBufIndex.Create

Synopsis:

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

Visibility: public

6.12.4 TArrayBufIndex.ScrollBackward

Synopsis:

Declaration: `function ScrollBackward : TGetResult;  Override`

Visibility: public

6.12.5 TArrayBufIndex.ScrollForward

Synopsis:

Declaration: `function ScrollForward : TGetResult;  Override`

Visibility: public

6.12.6 TArrayBufIndex.GetCurrent

Synopsis:

Declaration: `function GetCurrent : TGetResult;  Override`

Visibility: public

6.12.7 TArrayBufIndex.ScrollFirst

Synopsis:

Declaration: `function ScrollFirst : TGetResult;  Override`

Visibility: public

6.12.8 TArrayBufIndex.ScrollLast

Synopsis:

Declaration: `procedure ScrollLast;  Override`

Visibility: public

6.12.9 TArrayBufIndex.SetToFirstRecord

Synopsis:

Declaration: `procedure SetToFirstRecord;  Override`

Visibility: public

6.12.10 TArrayBufIndex.SetToLastRecord

Synopsis:

Declaration: `procedure SetToLastRecord;  Override`

Visibility: public

6.12.11 TArrayBufIndex.StoreCurrentRecord

Synopsis:

Declaration: `procedure StoreCurrentRecord;  Override`

Visibility: public

6.12.12 TArrayBufIndex.RestoreCurrentRecord

Synopsis:

Declaration: `procedure RestoreCurrentRecord;  Override`

Visibility: public

6.12.13 TArrayBufIndex.CanScrollForward

Synopsis:

Declaration: `function CanScrollForward : Boolean;  Override`

Visibility: public

6.12.14 TArrayBufIndex.DoScrollForward

Synopsis:

Declaration: `procedure DoScrollForward;  Override`

Visibility: public

6.12.15 TArrayBufIndex.StoreCurrentRecIntoBookmark

Synopsis:

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

Visibility: public

6.12.16 TArrayBufIndex.StoreSpareRecIntoBookmark

Synopsis:

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

Visibility: public

6.12.17 TArrayBufIndex.GotoBookmark

Synopsis:

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

Visibility: public

6.12.18 TArrayBufIndex.InitialiseIndex

Synopsis:

Declaration: `procedure InitialiseIndex;  Override`

Visibility: public

6.12.19 TArrayBufIndex.InitialiseSpareRecord

Synopsis:

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

Visibility: public

6.12.20 TArrayBufIndex.ReleaseSpareRecord

Synopsis:

Declaration: `procedure ReleaseSpareRecord;  Override`

Visibility: public

6.12.21 TArrayBufIndex.BeginUpdate

Synopsis:

Declaration: `procedure BeginUpdate;  Override`

Visibility: public

6.12.22 TArrayBufIndex.AddRecord

Synopsis:

Declaration: `procedure AddRecord;  Override`

Visibility: public

6.12.23 TArrayBufIndex.InsertRecordBeforeCurrentRecord

Synopsis:

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

Visibility: public

6.12.24 TArrayBufIndex.RemoveRecordFromIndex

Synopsis:

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

Visibility: public

6.12.25 TArrayBufIndex.EndUpdate

Synopsis:

Declaration: `procedure EndUpdate;  Override`

Visibility: public

6.13 TBufBlobStream

6.13.1 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 (173)

6.13.2 Method overview

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

6.13.3 TBufBlobStream.Create

Synopsis: Constructor for the class instance.

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

Visibility: public

Description: Constructor for the class instance.

6.13.4 TBufBlobStream.Destroy

Synopsis: Destructor for the class instance.

Declaration: `destructor Destroy;  Override`

Visibility: public

Description: Destroy is the destructor for the class instance.

6.14 TBufDataset

6.14.1 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 (166), TDataset (409), TDBDataset (455)

6.14.2 Property overview

Page	Properties	Access	Description
154	Active		
154	AfterCancel		
155	AfterClose		
155	AfterDelete		
155	AfterEdit		
155	AfterInsert		
155	AfterOpen		
155	AfterPost		
155	AfterScroll		
154	AutoCalcFields		
156	BeforeCancel		
156	BeforeClose		
156	BeforeDelete		
156	BeforeEdit		
156	BeforeInsert		
156	BeforeOpen		
156	BeforePost		
157	BeforeScroll		
153	FieldDefs		
154	Filter		
154	Filtered		
153	MaxIndexesCount		
157	OnCalcFields		
157	OnDeleteError		
157	OnEditError		
157	OnFilterRecord		
157	OnNewRecord		
157	OnPostError		
154	ReadOnly		

6.14.3 TBufDataset.MaxIndexesCount

Declaration: `Property MaxIndexesCount :`

Visibility: published

Access:

6.14.4 TBufDataset.FieldDefs

Declaration: `Property FieldDefs :`

Visibility: published

Access:

6.14.5 TBufDataset.Active

Declaration: `Property Active :`

Visibility: published

Access:

6.14.6 TBufDataset.AutoCalcFields

Declaration: `Property AutoCalcFields :`

Visibility: published

Access:

6.14.7 TBufDataset.Filter

Declaration: `Property Filter :`

Visibility: published

Access:

6.14.8 TBufDataset.Filtered

Declaration: `Property Filtered :`

Visibility: published

Access:

6.14.9 TBufDataset.ReadOnly

Declaration: `Property ReadOnly :`

Visibility: published

Access:

6.14.10 TBufDataset.AfterCancel

Declaration: `Property AfterCancel :`

Visibility: published

Access:

6.14.11 TBufDataset.AfterClose

Declaration: `Property AfterClose :`

Visibility: published

Access:

6.14.12 TBufDataset.AfterDelete

Declaration: `Property AfterDelete :`

Visibility: published

Access:

6.14.13 TBufDataset.AfterEdit

Declaration: `Property AfterEdit :`

Visibility: published

Access:

6.14.14 TBufDataset.AfterInsert

Declaration: `Property AfterInsert :`

Visibility: published

Access:

6.14.15 TBufDataset.AfterOpen

Declaration: `Property AfterOpen :`

Visibility: published

Access:

6.14.16 TBufDataset.AfterPost

Declaration: `Property AfterPost :`

Visibility: published

Access:

6.14.17 TBufDataset.AfterScroll

Declaration: `Property AfterScroll :`

Visibility: published

Access:

6.14.18 TBufDataset.BeforeCancel

Declaration: `Property BeforeCancel :`

Visibility: published

Access:

6.14.19 TBufDataset.BeforeClose

Declaration: `Property BeforeClose :`

Visibility: published

Access:

6.14.20 TBufDataset.BeforeDelete

Declaration: `Property BeforeDelete :`

Visibility: published

Access:

6.14.21 TBufDataset.BeforeEdit

Declaration: `Property BeforeEdit :`

Visibility: published

Access:

6.14.22 TBufDataset.BeforeInsert

Declaration: `Property BeforeInsert :`

Visibility: published

Access:

6.14.23 TBufDataset.BeforeOpen

Declaration: `Property BeforeOpen :`

Visibility: published

Access:

6.14.24 TBufDataset.BeforePost

Declaration: `Property BeforePost :`

Visibility: published

Access:

6.14.25 TBufDataset.BeforeScroll

Declaration: `Property BeforeScroll :`

Visibility: published

Access:

6.14.26 TBufDataset.OnCalcFields

Declaration: `Property OnCalcFields :`

Visibility: published

Access:

6.14.27 TBufDataset.OnDeleteError

Declaration: `Property OnDeleteError :`

Visibility: published

Access:

6.14.28 TBufDataset.OnEditError

Declaration: `Property OnEditError :`

Visibility: published

Access:

6.14.29 TBufDataset.OnFilterRecord

Declaration: `Property OnFilterRecord :`

Visibility: published

Access:

6.14.30 TBufDataset.OnNewRecord

Declaration: `Property OnNewRecord :`

Visibility: published

Access:

6.14.31 TBufDataset.OnPostError

Declaration: `Property OnPostError :`

Visibility: published

Access:

6.15 TBufIndex

6.15.1 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 (185), TUniDirectionalBufIndex (192), TArrayBufIndex (147)

6.15.2 Method overview

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

6.15.3 Property overview

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

6.15.4 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.

6.15.5 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 (362), TDataset.MoveBy (427)

6.15.6 TBufIndex.ScrollForward

Synopsis: Moves to the next record in the index.

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

Visibility: public

6.15.7 TBufIndex.GetCurrent

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

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

Visibility: public

6.15.8 TBufIndex.ScrollFirst

Synopsis: Moves to the first record in the index.

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

Visibility: public

6.15.9 TBufIndex.ScrollLast

Synopsis: Moves to the last record in the index.

Declaration: `procedure ScrollLast; Virtual; Abstract`

Visibility: public

6.15.10 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.

6.15.11 TBufIndex.SetToFirstRecord

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

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

Visibility: public

6.15.12 TBufIndex.SetToLastRecord

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

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

Visibility: public

6.15.13 TBufIndex.StoreCurrentRecord

Synopsis: Stores the record buffer for the current record.

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

Visibility: public

6.15.14 TBufIndex.RestoreCurrentRecord

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

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

Visibility: public

6.15.15 TBufIndex.CanScrollForward

Synopsis: Indicates if the index can be scrolled forward.

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

Visibility: public

6.15.16 TBufIndex.DoScrollForward

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

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

Visibility: public

6.15.17 TBufIndex.StoreCurrentRecIntoBookmark

Synopsis:

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

Visibility: public

6.15.18 TBufIndex.StoreSpareRecIntoBookmark

Synopsis:

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

Visibility: public

6.15.19 TBufIndex.GotoBookmark

Synopsis: Moves the index position to the specified Bookmark.

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

Visibility: public

6.15.20 TBufIndex.BookmarkValid

Synopsis: Indicates the specified Bookmark is valid.

Declaration:
```
function BookmarkValid(const ABookmark: PBufBookmark) : Boolean
                       ;  Virtual
```

Visibility: public

6.15.21 TBufIndex.CompareBookmarks

Synopsis: Gets the relative order for the specified Bookmarks.

Declaration:
```
function CompareBookmarks(const ABookmark1: PBufBookmark;
                          const ABookmark2: PBufBookmark) : Integer
                          ;  Virtual
```

Visibility: public

6.15.22 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

6.15.23 TBufIndex.InitialiseIndex

Synopsis: Initializes the index.

Declaration:
```
procedure InitialiseIndex;  Virtual;  Abstract
```

Visibility: public

Description: Initializes the index.

6.15.24 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.

6.15.25 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.

6.15.26 TBufIndex.BeginUpdate

Synopsis:

Declaration:
```
procedure BeginUpdate;  Virtual;  Abstract
```

Visibility: public

Description:

6.15.27 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.

6.15.28 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.

6.15.29 TBufIndex.RemoveRecordFromIndex

Synopsis: Remove the record at the specified bookmark from the index.

Declaration:
```
procedure RemoveRecordFromIndex(const ABookmark: TBufBookmark);  Virtual
                                ;  Abstract
```

Visibility: public

6.15.30 TBufIndex.OrderCurrentRecord

Synopsis:

Declaration:
```
procedure OrderCurrentRecord;  Virtual;  Abstract
```

Visibility: public

Description:

6.15.31 TBufIndex.EndUpdate

Synopsis:

Declaration:
```
procedure EndUpdate;  Virtual;  Abstract
```

Visibility: public

Description:

6.15.32 TBufIndex.SpareRecord

Synopsis: Spare record for the index.

Declaration: `Property SpareRecord : TRecordBuffer`

Visibility: public

Access: Read

Description: Spare record for the index.

6.15.33 TBufIndex.SpareBuffer

Synopsis: Spare record buffer for the index.

Declaration: `Property SpareBuffer : TRecordBuffer`

Visibility: public

Access: Read

Description: Spare record buffer for the index.

6.15.34 TBufIndex.CurrentRecord

Synopsis: Current record in the index.

Declaration: `Property CurrentRecord : TRecordBuffer`

Visibility: public

Access: Read

Description: Current record in the index.

6.15.35 TBufIndex.CurrentBuffer

Synopsis: Current record buffer in the index.

Declaration: `Property CurrentBuffer : Pointer`

Visibility: public

Access: Read

Description: Current record buffer in the index.

6.15.36 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.

6.15.37 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.

6.15.38 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.

6.16 TCustomBufDataset

6.16.1 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 (152), TDBDataset (455), TDataset (409)

6.16.2 Method overview

Page	Method	Description
173	AddIndex	Adds an index definition to the dataset.
169	ApplyUpdates	Applies pending updates to the dataset.
178	BookmarkValid	Determines if the specified Bookmark is valid for the dataset.
170	CancelUpdates	Cancels pending updates in the dataset.
177	Clear	Clears the content in the dataset.
174	ClearIndexes	Clears index storage in the dataset.
178	CompareBookmarks	Gets the relative sort order for the specified Bookmarks.
178	CopyFromDataset	Loads field definitions and optional data from another dataset.
168	Create	Constructor for the class instance.
173	CreateBlobStream	Creates a Blob stream for the specified field with the given permissions.
177	CreateDataset	Creates the dataset using its field definitions or bound fields.
170	Destroy	Destructor for the class instance.
174	GetDatasetPacket	Builds a data packet representing the content in the buffered dataset.
169	GetFieldData	
176	LoadFromFile	Loads the dataset from the specified file using the given format.
175	LoadFromStream	Loads the dataset from the specified stream using the given data format.
171	Locate	Locates the first record with fields having the specified values.
171	Lookup	Gets values from the first record with fields that match the search condition.
170	MergeChangeLog	Frees update buffers and Blob update buffer allocated for the dataset.
170	RevertRecord	Reverts the current record to its original (un-edited) values.
176	SaveToFile	Saves the dataset to the specified file using the given data format.
175	SaveToStream	Saves the dataset to the specified stream using the given data format.
174	SetDatasetPacket	Retrieves and applies the data packet with the content for the dataset.
169	SetFieldData	
172	UpdateStatus	Gets the update status for the current record in the dataset.

6.16.3 Property overview

Page	Properties	Access	Description
179	ChangeCount	r	Number of pending changes for the dataset.
180	FileName	rw	File name on the local file system used to load or store the dataset.
182	IndexDefs	r	Index definitions for the dataset.
182	IndexFieldNames	rw	Field names included in the custom index.
182	IndexName	rw	Name of the selected index for the dataset.
180	ManualMergeChangeLog	rw	Indicates if the update change log can be manually merged.
179	MaxIndexesCount	rw	Maximum number of indexes available in the dataset.
181	OnUpdateError	rw	Event handler signalled when an error occurs while updating records.
181	PacketRecords	rw	Number of records allowed in a data packet handled by the packet reader.
180	ReadOnly	rw	Indicates if records can be added, deleted, or modified in the dataset.
183	UniDirectional	rw	Indicates if the dataset is for uni-directional navigation only.

6.16.4 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 (179), TCustomBufDataset.PacketRecords (181)

6.16.5 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:

6.16.6 TCustomBufDataset.SetFieldData

Synopsis:

Declaration:
```
procedure SetFieldData(Field: TField; Buffer: Pointer;
                       NativeFormat: Boolean);  Override
procedure SetFieldData(Field: TField; Buffer: Pointer);  Override
```

Visibility: public

Description:

6.16.7 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 (417), TRecUpdateBuffer (147), TCustomBufDataset.OnUpdateError (181), TCustomBufDataset.ManualMergeChangeLog (180), TCustomBufDataset.MergeChangeLog (170), TDataset.Resync (429)

6.16.8 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 (169)

6.16.9 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 (417), TDataset.Active (439), TCustomBufDataset.Resync (166)

6.16.10 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 (417), TDataset.Active (439), TCustomBufDataset.Resync (166)

6.16.11 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.

6.16.12 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 (426), TCustomBufDataset.UniDirectional (183), TDataset.Active (439), TDataset.Filter (438), TDataset.OnFilterRecord (438), TLocateOptions (363), TDBCompareStruct (143)

6.16.13 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 (171), TDataset.Active (439), TCustomBufDataset.UniDirectional (183)

6.16.14 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:

usUnmodifiedRecord has not been modified

usModifiedRecord has been modified

usInsertedRecords was appended or inserted

usDeletedRecord 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 (368), TDataset.ActiveRecord (409)

6.16.15 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 (419), TField (462), TBlobStreamMode (354)

6.16.16 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 (183), TDataset.Active (439), TBufDatasetIndex (142)

6.16.17 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 (409), TCustomBufDataset.UniDirectional (183)

6.16.18 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 (428), TCustomBufDataset.LoadFromStream (175), TDataPacketReader (143)

6.16.19 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 (143), TCustomBufDataset.SaveToStream (175)

6.16.20 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 (183), TCustomBufDataset.SetDatasetPacket (174), TCustomBufDataset.LoadFromFile (176), TDataset.Open (428), TDataPacketFormat (143)

6.16.21 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 (183), TCustomBufDataset.SaveToFile (176), TDataPacketFormat (143), RegisterDatapacketReader (145), TDataPacketReaderClass (143), TDataPacketReader (143)

6.16.22 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 (180), TCustomBufDataset.LoadFromStream (175)

6.16.23 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 (180), TCustomBufDataset.SaveToStream (175)

6.16.24 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 (439), TDataset.FieldDefs (434), TDataset.Fields (437), TDataset.Open (428), TCustomBufDataset.FileName (180)

6.16.25 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 (418), TDataset.FieldDefs (434), TDataset.Fields (437)

6.16.26 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 (162)

6.16.27 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 (158)

6.16.28 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 (418), TDataset.Cancel (417), TDataset.FieldDefs (434)

6.16.29 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 (169), TCustomBufDataset.CancelUpdates (170), TCustomBufDataset.RevertRecord (170)

6.16.30 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 (439), TCustomBufDataset.AddIndex (173)

6.16.31 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 (183), TDataset.CanModify (432), TDataset.FieldDefs (434), TDataset.Fields (437)

6.16.32 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`.

6.16.33 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 (176), TCustomBufDataset.SaveToFile (176), TDataset.Open (428), TDataset.Close (418)

6.16.34 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.

6.16.35 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 (144), TCustomBufDataset.ApplyUpdates (169)

6.16.36 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 (166), TIndexDefs (512)

6.16.37 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 (439), TDataset.Resync (429)

6.16.38 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 (166), TDataset.Active (439)

6.16.39 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 (409), TDataset.Active (439)

6.17 TDataPacketHandler

6.17.1 Method overview

Page	Method	Description
184	Create	
185	FinalizeStoreRecords	
184	GetCurrentRecord	
184	GetRecordRowState	
184	GotoNextRecord	
184	InitLoadRecords	
184	LoadFieldDefs	
185	RecognizeStream	
184	RestoreRecord	
184	StoreFieldDefs	
185	StoreRecord	

6.17.2 TDataPacketHandler.Create

Declaration: `constructor Create(ADataSet: TCustomBufDataset; AStream: TStream) ; Virtual`

Visibility: public

6.17.3 TDataPacketHandler.LoadFieldDefs

Declaration: `procedure LoadFieldDefs(var AnAutoIncValue: Integer); Virtual ; Abstract`

Visibility: public

6.17.4 TDataPacketHandler.InitLoadRecords

Declaration: `procedure InitLoadRecords; Virtual; Abstract`

Visibility: public

6.17.5 TDataPacketHandler.GetCurrentRecord

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

Visibility: public

6.17.6 TDataPacketHandler.GetRecordRowState

Declaration: `function GetRecordRowState(out AUpdOrder: Integer) : TRowState; Virtual ; Abstract`

Visibility: public

6.17.7 TDataPacketHandler.RestoreRecord

Declaration: `procedure RestoreRecord; Virtual; Abstract`

Visibility: public

6.17.8 TDataPacketHandler.GotoNextRecord

Declaration: `procedure GotoNextRecord; Virtual; Abstract`

Visibility: public

6.17.9 TDataPacketHandler.StoreFieldDefs

Declaration: `procedure StoreFieldDefs(AnAutoIncValue: Integer); Virtual; Abstract`

Visibility: public

6.17.10 TDataPacketHandler.StoreRecord

Declaration: `procedure StoreRecord(ARowState: TRowState; AUpdOrder: Integer); Virtual; Abstract`

Visibility: public

6.17.11 TDataPacketHandler.FinalizeStoreRecords

Declaration: `procedure FinalizeStoreRecords; Virtual; Abstract`

Visibility: public

6.17.12 TDataPacketHandler.RecognizeStream

Declaration: `class function RecognizeStream(AStream: TStream) : Boolean; Virtual; Abstract`

Visibility: public

6.18 TDoubleLinkedBufIndex

6.18.1 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 (158), TBufRecLinkItem (146), PBufRecLinkItem (143), TBufDataset (152)

6.18.2 Method overview

Page	Method	Description
190	AddRecord	Adds a record to the index.
189	BeginUpdate	
188	CanScrollForward	Indicates if the index can be scrolled towards the end of the index.
189	CompareBookmarks	
188	DoScrollForward	Implements actions needed to scroll forward using the index.
190	EndUpdate	
186	GetCurrent	
187	GetRecord	
188	GotoBookmark	Navigates the index to the specified Bookmark.
189	InitialiseIndex	Initializes the index.
189	InitialiseSpareRecord	Initializes the spare record in the index.
190	InsertRecordBeforeCurrentRecord	Inserts a record prior to the current record in the index.
190	OrderCurrentRecord	
189	ReleaseSpareRecord	Releases the spare record in the index.
190	RemoveRecordFromIndex	Removes the record at the specified Bookmark from the index.
188	RestoreCurrentRecord	
189	SameBookmarks	Compares Bookmark values for ordering in the index.
186	ScrollBackward	Scrolls the index toward the beginning of the index.
187	ScrollFirst	Scrolls to the first entry in the index.
186	ScrollForward	Scrolls the index towards the end of the index.
187	ScrollLast	Scrolls to the last entry in the index.
187	SetToFirstRecord	
187	SetToLastRecord	
188	StoreCurrentRecIntoBookmark	
187	StoreCurrentRecord	
188	StoreSpareRecIntoBookmark	

6.18.3 TDoubleLinkedBufIndex.ScrollBackward

Synopsis: Scrolls the index toward the beginning of the index.

Declaration: `function ScrollBackward : TGetResult;  Override`

Visibility: public

6.18.4 TDoubleLinkedBufIndex.ScrollForward

Synopsis: Scrolls the index towards the end of the index.

Declaration: `function ScrollForward : TGetResult;  Override`

Visibility: public

6.18.5 TDoubleLinkedBufIndex.GetCurrent

Synopsis:

Declaration: `function GetCurrent : TGetResult;  Override`

Visibility: public

6.18.6 TDoubleLinkedBufIndex.ScrollFirst

Synopsis: Scrolls to the first entry in the index.

Declaration: `function ScrollFirst : TGetResult;  Override`

Visibility: public

6.18.7 TDoubleLinkedBufIndex.ScrollLast

Synopsis: Scrolls to the last entry in the index.

Declaration: `procedure ScrollLast;  Override`

Visibility: public

6.18.8 TDoubleLinkedBufIndex.GetRecord

Synopsis:

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

Visibility: public

6.18.9 TDoubleLinkedBufIndex.SetToFirstRecord

Synopsis:

Declaration: `procedure SetToFirstRecord;  Override`

Visibility: public

6.18.10 TDoubleLinkedBufIndex.SetToLastRecord

Synopsis:

Declaration: `procedure SetToLastRecord;  Override`

Visibility: public

6.18.11 TDoubleLinkedBufIndex.StoreCurrentRecord

Synopsis:

Declaration: `procedure StoreCurrentRecord;  Override`

Visibility: public

6.18.12 TDoubleLinkedBufIndex.RestoreCurrentRecord

Synopsis:

Declaration: `procedure RestoreCurrentRecord;  Override`

Visibility: public

6.18.13 TDoubleLinkedBufIndex.CanScrollForward

Synopsis: Indicates if the index can be scrolled towards the end of the index.

Declaration: `function CanScrollForward : Boolean;  Override`

Visibility: public

6.18.14 TDoubleLinkedBufIndex.DoScrollForward

Synopsis: Implements actions needed to scroll forward using the index.

Declaration: `procedure DoScrollForward;  Override`

Visibility: public

6.18.15 TDoubleLinkedBufIndex.StoreCurrentRecIntoBookmark

Synopsis:

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

Visibility: public

6.18.16 TDoubleLinkedBufIndex.StoreSpareRecIntoBookmark

Synopsis:

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

Visibility: public

6.18.17 TDoubleLinkedBufIndex.GotoBookmark

Synopsis: Navigates the index to the specified Bookmark.

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

Visibility: public

6.18.18 TDoubleLinkedBufIndex.CompareBookmarks

Synopsis:

Declaration:
```
function CompareBookmarks(const ABookmark1: PBufBookmark;
                          const ABookmark2: PBufBookmark) : Integer
                          ;  Override
```

Visibility: public

6.18.19 TDoubleLinkedBufIndex.SameBookmarks

Synopsis: Compares Bookmark values for ordering in the index.

Declaration:
```
function SameBookmarks(const ABookmark1: PBufBookmark;
                       const ABookmark2: PBufBookmark) : Boolean
                       ;  Override
```

Visibility: public

6.18.20 TDoubleLinkedBufIndex.InitialiseIndex

Synopsis: Initializes the index.

Declaration:
```
procedure InitialiseIndex;  Override
```

Visibility: public

6.18.21 TDoubleLinkedBufIndex.InitialiseSpareRecord

Synopsis: Initializes the spare record in the index.

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

Visibility: public

6.18.22 TDoubleLinkedBufIndex.ReleaseSpareRecord

Synopsis: Releases the spare record in the index.

Declaration:
```
procedure ReleaseSpareRecord;  Override
```

Visibility: public

6.18.23 TDoubleLinkedBufIndex.BeginUpdate

Synopsis:

Declaration:
```
procedure BeginUpdate;  Override
```

Visibility: public

6.18.24 TDoubleLinkedBufIndex.AddRecord

Synopsis: Adds a record to the index.

Declaration:
```
procedure AddRecord;  Override
```

Visibility: public

6.18.25 TDoubleLinkedBufIndex.InsertRecordBeforeCurrentRecord

Synopsis: Inserts a record prior to the current record in the index.

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

Visibility: public

6.18.26 TDoubleLinkedBufIndex.RemoveRecordFromIndex

Synopsis: Removes the record at the specified Bookmark from the index.

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

Visibility: public

6.18.27 TDoubleLinkedBufIndex.OrderCurrentRecord

Synopsis:

Declaration:
```
procedure OrderCurrentRecord;  Override
```

Visibility: public

6.18.28 TDoubleLinkedBufIndex.EndUpdate

Synopsis:

Declaration:
```
procedure EndUpdate;  Override
```

Visibility: public

6.19 TFpcBinaryDatapacketHandler

6.19.1 Method overview

Page	Method	Description
191	Create	
192	FinalizeStoreRecords	
191	GetCurrentRecord	
191	GetRecordRowState	
192	GotoNextRecord	
191	InitLoadRecords	
191	LoadFieldDefs	
192	RecognizeStream	
192	RestoreRecord	
191	StoreFieldDefs	
192	StoreRecord	

6.19.2 TFpcBinaryDatapacketHandler.Create

Declaration:
```
constructor Create(ADataSet: TCustomBufDataset; AStream: TStream)
                        ;  Override
```

Visibility: public

6.19.3 TFpcBinaryDatapacketHandler.LoadFieldDefs

Declaration:
```
procedure LoadFieldDefs(var AnAutoIncValue: Integer);  Override
```

Visibility: public

6.19.4 TFpcBinaryDatapacketHandler.StoreFieldDefs

Declaration:
```
procedure StoreFieldDefs(AnAutoIncValue: Integer);  Override
```

Visibility: public

6.19.5 TFpcBinaryDatapacketHandler.InitLoadRecords

Declaration:
```
procedure InitLoadRecords;  Override
```

Visibility: public

6.19.6 TFpcBinaryDatapacketHandler.GetCurrentRecord

Declaration:
```
function GetCurrentRecord : Boolean;  Override
```

Visibility: public

6.19.7 TFpcBinaryDatapacketHandler.GetRecordRowState

Declaration:
```
function GetRecordRowState(out AUpdOrder: Integer) : TRowState
                                ;  Override
```

Visibility: public

6.19.8 TFpcBinaryDatapacketHandler.RestoreRecord

Declaration: `procedure RestoreRecord;  Override`

Visibility: public

6.19.9 TFpcBinaryDatapacketHandler.GotoNextRecord

Declaration: `procedure GotoNextRecord;  Override`

Visibility: public

6.19.10 TFpcBinaryDatapacketHandler.StoreRecord

Declaration:
```
procedure StoreRecord(ARowState: TRowState; AUpdOrder: Integer)
                      ;  Override
```

Visibility: public

6.19.11 TFpcBinaryDatapacketHandler.FinalizeStoreRecords

Declaration: `procedure FinalizeStoreRecords;  Override`

Visibility: public

6.19.12 TFpcBinaryDatapacketHandler.RecognizeStream

Declaration: `class function RecognizeStream(AStream: TStream) : Boolean;  Override`

Visibility: public

6.20 TUniDirectionalBufIndex

6.20.1 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 (158)

6.20.2 Method overview

Page	Method	Description
196	AddRecord	
196	BeginUpdate	
194	CanScrollForward	
195	DoScrollForward	
196	EndUpdate	
193	GetCurrent	
195	GotoBookmark	
195	InitialiseIndex	
195	InitialiseSpareRecord	
196	InsertRecordBeforeCurrentRecord	
196	OrderCurrentRecord	
195	ReleaseSpareRecord	
196	RemoveRecordFromIndex	
194	RestoreCurrentRecord	
193	ScrollBackward	
193	ScrollFirst	
193	ScrollForward	
194	ScrollLast	
194	SetToFirstRecord	
194	SetToLastRecord	
195	StoreCurrentRecIntoBookmark	
194	StoreCurrentRecord	
195	StoreSpareRecIntoBookmark	

6.20.3 TUniDirectionalBufIndex.ScrollBackward

Synopsis:

Declaration: `function ScrollBackward : TGetResult;  Override`

Visibility: public

6.20.4 TUniDirectionalBufIndex.ScrollForward

Synopsis:

Declaration: `function ScrollForward : TGetResult;  Override`

Visibility: public

6.20.5 TUniDirectionalBufIndex.GetCurrent

Synopsis:

Declaration: `function GetCurrent : TGetResult;  Override`

Visibility: public

6.20.6 TUniDirectionalBufIndex.ScrollFirst

Synopsis:

Declaration: `function ScrollFirst : TGetResult;  Override`

Visibility: public

6.20.7 TUniDirectionalBufIndex.ScrollLast

Synopsis:

Declaration: `procedure ScrollLast;  Override`

Visibility: public

6.20.8 TUniDirectionalBufIndex.SetToFirstRecord

Synopsis:

Declaration: `procedure SetToFirstRecord;  Override`

Visibility: public

6.20.9 TUniDirectionalBufIndex.SetToLastRecord

Synopsis:

Declaration: `procedure SetToLastRecord;  Override`

Visibility: public

6.20.10 TUniDirectionalBufIndex.StoreCurrentRecord

Synopsis:

Declaration: `procedure StoreCurrentRecord;  Override`

Visibility: public

6.20.11 TUniDirectionalBufIndex.RestoreCurrentRecord

Synopsis:

Declaration: `procedure RestoreCurrentRecord;  Override`

Visibility: public

6.20.12 TUniDirectionalBufIndex.CanScrollForward

Synopsis:

Declaration: `function CanScrollForward : Boolean;  Override`

Visibility: public

6.20.13 TUniDirectionalBufIndex.DoScrollForward

Synopsis:

Declaration:
```
procedure DoScrollForward;  Override
```

Visibility: public

6.20.14 TUniDirectionalBufIndex.StoreCurrentRecIntoBookmark

Synopsis:

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

Visibility: public

6.20.15 TUniDirectionalBufIndex.StoreSpareRecIntoBookmark

Synopsis:

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

Visibility: public

6.20.16 TUniDirectionalBufIndex.GotoBookmark

Synopsis:

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

Visibility: public

6.20.17 TUniDirectionalBufIndex.InitialiseIndex

Synopsis:

Declaration:
```
procedure InitialiseIndex;  Override
```

Visibility: public

6.20.18 TUniDirectionalBufIndex.InitialiseSpareRecord

Synopsis:

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

Visibility: public

6.20.19 TUniDirectionalBufIndex.ReleaseSpareRecord

Synopsis:

Declaration:
```
procedure ReleaseSpareRecord;  Override
```

Visibility: public

6.20.20 TUniDirectionalBufIndex.BeginUpdate

Synopsis:

Declaration: `procedure BeginUpdate;  Override`

Visibility: public

6.20.21 TUniDirectionalBufIndex.AddRecord

Synopsis:

Declaration: `procedure AddRecord;  Override`

Visibility: public

6.20.22 TUniDirectionalBufIndex.InsertRecordBeforeCurrentRecord

Synopsis:

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

Visibility: public

6.20.23 TUniDirectionalBufIndex.RemoveRecordFromIndex

Synopsis:

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

Visibility: public

6.20.24 TUniDirectionalBufIndex.OrderCurrentRecord

Synopsis:

Declaration: `procedure OrderCurrentRecord;  Override`

Visibility: public

6.20.25 TUniDirectionalBufIndex.EndUpdate

Synopsis:

Declaration: `procedure EndUpdate;  Override`

Visibility: public

Chapter 7

Reference for unit 'bufstream'

7.1 Used units

Table 7.1: Used units by unit 'bufstream'

Name	Page
Classes	??
System	??
sysutils	??

7.2 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 (203) is used for reading only, and allows the buffer size to be specified at the time of creation.

TWriteBufStream (204) is used for writing only, and allows the buffer size to be specified at the time of creation.

TBufferedFileStream (198) 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.

7.3 Constants, types and variables

7.3.1 Constants

```
DefaultBufferCapacity : Integer = 16
```

If no buffer size is specified when the stream is created, then this size is used.

7.4 TBufferedFileStream

7.4.1 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 (201) 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 (201).

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 (203) or TWriteBufStream (204) for speeding up sequential writes.

See also: TFileStream (**??**), THandleStream (**??**), TStream (**??**), TReadBufStream (203), TWriteBufStream (204)

7.4.2 Method overview

Page	Method	Description
198	Create	Constructor for the class instance.
199	Destroy	Destructor for the class instance.
201	Flush	Flushes modified pages in the buffer to the file stream.
201	InitializeCache	Re-initializes the internal buffer for the buffered file stream.
200	Read	Reads the specified number of bytes into the Buffer parameter.
199	Seek	Moves the position in the buffer relative to the specified origin.
200	Write	Writes the specified number of bytes in Buffer to the internal page buffer(s).

7.4.3 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:

fmCreateCreates the file it does not already exist.

fmOpenReadOpens the file for read-only access.

fmOpenWriteOpens the file for write-only access.

fmOpenReadWriteOpens the file for read / write access.

The file mode constants (**except for fmCreate**) can be **OR**'d with sharing mode constants, including:

fmShareCompatOpens the file in DOS-compatibility sharing mode.

fmShareExclusiveLocks the file for exclusive use.

fmShareDenyWriteLocks the file and denies write access to other processes.

fmShareDenyReadLocks the file and denies read access to other processes.

fmShareDenyNoneDoes 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 (201), TBufferedFileStream.Size (198), TFileStream.Create (**??**), TFileStream.Size (**??**)

7.4.4 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 (**??**)

7.4.5 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 (**??**)

7.4.6 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 (**??**)

7.4.7 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 (199), TBufferedFileStream.GetPosition (198), TBufferedFileStream.SetPosition (198), THandleStream.Write (**??**), TStream.Write (**??**), TStream.Position (**??**), EStreamError (**??**)

7.4.8 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 (200), THandleStream.Seek (**??**), THandleStream.Write (**??**), TStream.Seek (**??**), TStream.Write (**??**), EStreamError (**??**)

7.4.9 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 (201), TBufferedFileStream.Create (198)

7.5 TBufStream

7.5.1 Description

`TBufStream` is the common ancestor for the TReadBufStream (203) and TWriteBufStream (204) 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 (203), TWriteBufStream (204)

7.5.2 Method overview

Page	Method	Description
202	Create	Create a new `TBufStream` instance.
202	Destroy	Destroys the `TBufStream` instance.

7.5.3 Property overview

Page	Properties	Access	Description
202	Buffer	r	The current buffer.
203	BufferPos	r	Current buffer position.
203	BufferSize	r	Amount of data in the buffer.
203	Capacity	rw	Current buffer capacity.

7.5.4 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 (197) is used as the capacity.

An instance of `TBufStream` should never be instantiated directly. Instead, an instance of TReadBufStream (203) or TWriteBufStream (204) should be created.

Errors: If not enough memory is available for the buffer, then an exception may be raised.

See also: TBufStream.Destroy (202), TReadBufStream (203), TWriteBufStream (204)

7.5.5 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 (202), TReadBufStream (203), TWriteBufStream (204)

7.5.6 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 (202), TBufStream.Capacity (203), TBufStream.BufferSize (203)

7.5.7 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 (202), TBufStream.Buffer (202), TBufStream.BufferSize (203), TBufStream.BufferPos (203)

7.5.8 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 (202), TBufStream.Buffer (202), TBufStream.BufferSize (203), TBufStream.Capacity (203)

7.5.9 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 (203).

See also: TBufStream.Create (202), TBufStream.Buffer (202), TBufStream.BufferPos (203), TBufStream.Capacity (203)

7.6 TReadBufStream

7.6.1 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 (201), TWriteBufStream (204)

7.6.2 Method overview

Page	Method	Description
204	Read	Reads data from the stream.
204	Seek	Set location in the buffer.

7.6.3 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 (205), TReadBufStream.Read (204)

7.6.4 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 (204), TReadBufStream.Read (204)

7.7 TWriteBufStream

7.7.1 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 (201), TReadBufStream (203)

7.7.2 Method overview

Page	Method	Description
205	Destroy	Remove the `TWriteBufStream` instance from memory.
205	Seek	Set stream position.
205	Write	Write data to the stream.

7.7.3 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 (202).

Errors: If an error occurs during flushing of the buffer, an exception may be raised.

See also: Create (202), TBufStream.Destroy (202)

7.7.4 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 (205), TReadBufStream.Seek (204)

7.7.5 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 (205)

Chapter 8

Reference for unit 'CacheCls'

8.1 Used units

Table 8.1: Used units by unit 'CacheCls'

Name	Page
System	??
sysutils	??

8.2 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`

8.3 Constants, types and variables

8.3.1 Resource strings

```
SInvalidIndex = 'Invalid index %i'
```

Message shown when an invalid index is passed.

8.3.2 Types

```
PCacheSlot = ^TCacheSlot
```

Pointer to TCacheSlot (207) record.

```
PCacheSlotArray = ^TCacheSlotArray
```

Pointer to TCacheSlotArray (207) array.

```
TCacheSlotArray = Array[0..MaxIntdivSizeOf(TCacheSlot)-1] of
  TCacheSlot
```

TCacheSlotArray is an array of TCacheSlot items. Do not use TCacheSlotArray directly, instead, use PCacheSlotArray (206) 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 (208) 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.

8.4 TCacheSlot

```
TCacheSlot = record
  Prev : PCacheSlot;
  Next : PCacheSlot;
  Data
  : Pointer;
  Index : Integer;
end
```

TCacheSlot is internally used by the TCache (207) class. It represents 1 element in the linked list.

8.5 ECacheError

8.5.1 Description

Exception class used in the cachecls unit.

8.6 TCache

8.6.1 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 (210) property offers indexed access to the array of items. When accessing the array through this property, the MRUSlot (210) property is updated.

8.6.2 Method overview

Page	Method	Description
208	Add	Add a data element to the list.
209	AddNew	Add a new item to the list.
208	Create	Create a new cache class.
208	Destroy	Free the `TCache` class from memory.
209	FindSlot	Find data pointer in the list.
209	IndexOf	Return index of a data pointer in the list.
210	Remove	Remove a data item from the list.

8.6.3 Property overview

Page	Properties	Access	Description
210	Data	rw	Indexed access to data items.
211	LRUSlot	r	Last used item.
210	MRUSlot	rw	Most recent item slot.
212	OnFreeSlot	rw	Event called when a slot is freed.
211	OnIsDataEqual	rw	Event to compare 2 items.
211	SlotCount	rw	Number of slots in the list.
211	Slots	r	Indexed array to the slots.

8.6.4 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 (211)

8.6.5 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 (208)

8.6.6 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 (209) 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 (209), TCache.FindSlot (209), TCache.IndexOf (209), TCache.Data (210), TCache.MRUSlot (210)

8.6.7 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 (208) 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 (208), TCache.FindSlot (209), TCache.IndexOf (209), TCache.Data (210), TCache.MRUSlot (210)

8.6.8 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 (211) event must be set.

Errors: If `OnIsDataEqual` is not set, an exception will be raised.

See also: TCache.IndexOf (209), TCache.Add (208), TCache.OnIsDataEqual (211)

8.6.9 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 (211) event must be set.

Errors: If `OnIsDataEqual` is not set, an exception will be raised.

See also: TCache.FindSlot (209), TCache.Add (208), TCache.OnIsDataEqual (211)

8.6.10 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 (209)

8.6.11 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 (210) will point to the accessed item. The access is both read and write.

The index is zero-based and can maximally be SlotCount-1 (211). Providing an invalid index will result in an exception.

See also: TCache.MRUSlot (210)

8.6.12 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 (210) property, or when an item is added to the list with Add (208) or AddNew (209)

See also: TCache.Add (208), TCache.AddNew (209), TCache.Data (210), TCache.LRUSlot (211)

8.6.13 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 (208), TCache.AddNew (209), TCache.Data (210), TCache.MRUSlot (210)

8.6.14 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 (212) is called.

See also: TCache.Create (208), TCache.Data (210), TCache.Slots (211)

8.6.15 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 (211). Providing an invalid index will result in an exception.

See also: TCache.Data (210), TCache.SlotCount (211)

8.6.16 TCache.OnIsDataEqual

Synopsis: Event to compare 2 items.

Declaration: `Property OnIsDataEqual : TOnIsDataEqual`

Visibility: public

Access: Read,Write

Description: `OnIsDataEqual` is used by FindSlot (209) and IndexOf (209) to compare items when looking for a particular item. These functions are called by the Add (208) 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 (209), TCache.IndexOf (209), TCache.Add (208)

8.6.17 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 (208), TCache.AddNew (209), TCache.SlotCount (211)

Chapter 9

Reference for unit 'Contnrs'

9.1 Used units

Table 9.1: Used units by unit 'Contnrs'

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

9.2 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.

9.3 Constants, types and variables

9.3.1 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.

9.3.2 Types

```
PBucket = ^TBucket
```

Pointer to TBucket (217)" type.

```
PHashItem = ^THashItem
```

PHashItem is a pointer type, pointing to the THashItem (218) record.

```
PHashItemList = ^THashItemList
```

PHashItemList is a pointer to the THashItemList (215). It's used in the TFPHashList (234) as a pointer to the memory area containing the hash item records.

```
PHashTable = ^THashTable
```

PHashTable is a pointer to the THashTable (215). It's used in the TFPHashList (234) as a pointer to the memory area containing the hash values.

```
TBucketArray = Array of TBucket
```

Array of TBucket (217) records.

```
TBucketItemArray = Array of TBucketItem
```

Array of TBucketItem records.

```
TBucketListSizes = (bl2,bl4,bl8,bl16,bl32,bl64,bl128,bl256)
```

Table 9.2: 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 (218).

```
TBucketProc = procedure(AInfo: Pointer; AItem: Pointer; AData: Pointer
  ;
                        out AContinue: Boolean)
```

TBucketProc is the prototype for the TCustomBucketList.Foreach (227) 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 (215) prototype.

```
TBucketProcObject = procedure(AItem: Pointer; AData: Pointer;
  out AContinue: Boolean) of object
```

TBucketProcObject is the prototype for the TCustomBucketList.Foreach (227) 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 (215) prototype.

```
TDataIteratorCallBack = procedure(Item: Pointer; const Key: string
  ;
                                  var Continue: Boolean)
```

TDataIteratorCallBack is a callback prototype for the TFPDataHashTable.Iterate (233) 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: string;
  var Continue: Boolean) of
  object
```

TDataIteratorMethod is a callback prototype for the TFPDataHashTable.Iterate (233) 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: string; 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.

```
THashItemList = Array[0..MaxHashListSize-1] of THashItem
```

THashItemList is an array type, primarily used to be able to define the PHashItemList (214) type. It's used in the TFPHashList (234) class.

```
THashTable = Array[0..MaxHashTableSize-1] of Integer
```

THashTable defines an array of integers, used to hold hash values. It's mainly used to define the PHashTable (214) class.

```
THTCustomNodeClass = Class of THTCustomNode
```

THTCustomNodeClass was used by TFPCustomHashTable (227) 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 (233) method.

```
TObjectIteratorCallback = procedure(Item: TObject; const Key: string
  ;
                                    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: string
  ;
                                  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 (259) 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 (259) 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: string; const Key: string
  ;
                                    var Continue: Boolean)
```

TStringIteratorCallBack is the callback prototype for the TFPStringHashTable (261) 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: string; const Key: string
  ;
                                  var Continue: Boolean) of
  object
```

`TStringIteratorMethod` is the callback prototype for the TFPStringHashTable (261) method. It is called for each element in the hash table, with the string. If `Continue` is set to `False`, the iteration stops.

9.4 Procedures and functions

9.4.1 RSHash

Synopsis: Standard hash value calculating function.

Declaration: `function RSHash(const S: string; const TableSize: LongWord) : LongWord`

Visibility: default

Description: `RSHash` is the standard hash calculating function used in the TFPCustomHashTable (227) hash class. It's Robert Sedgwick's "Algorithms in C" hash function.

Errors: None.

See also: TFPCustomHashTable (227)

9.5 TBucket

```
TBucket = record
  Count : Integer;
  Items : TBucketItemArray;
end
```

`TBucket` describes 1 bucket in the TCustomBucketList (224) class. It is a container for TBucketItem (217) records. It should never be used directly.

9.6 TBucketItem

```
TBucketItem = record
  Item : Pointer;
  Data : Pointer;
end
```

`TBucketItem` is a record used for internal use in TCustomBucketList (224). It should not be necessary to use it directly.

9.7 THashItem

```
THashItem = record
  HashValue : LongWord;
  StrIndex : Integer;
  NextIndex : Integer;
  Data : Pointer;
end
```

THashItem is used internally in the hash list. It should never be used directly.

9.8 EDuplicate

9.8.1 Description

Exception raised when a key is stored twice in a hash table.

9.9 EKeyNotFound

9.9.1 Description

Exception raised when a key is not found.

See also: TFPCustomHashTable.Delete (230)

9.10 TBucketList

9.10.1 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 (224)

9.10.2 Method overview

Page	Method	Description
218	Create	Create a new TBucketList instance.

9.10.3 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 (214)

9.11 TClassList

9.11.1 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 (221), TObjectList (266)

9.11.2 Method overview

Page	Method	Description
219	Add	Add a new class pointer to the list.
220	Extract	Extract a class pointer from the list.
220	First	Returns the first non-nil class pointer.
220	IndexOf	Search for a class pointer in the list.
221	Insert	Insert a new class pointer in the list.
221	Last	Return last non-`Nil` class pointer.
220	Remove	Remove a class pointer from the list.

9.11.3 Property overview

Page	Properties	Access	Description
221	Items	rw	Index based access to class pointers.

9.11.4 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 (220), #rtl.classes.tlist.add (**??**)

9.11.5 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 (220), #rtl.classes.Tlist.Extract (**??**)

9.11.6 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 (220), #rtl.classes.Tlist.Remove (**??**)

9.11.7 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 (**??**)

9.11.8 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 (221)

9.11.9 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 (220)

9.11.10 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 (219), TClassList.Remove (220)

9.11.11 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 (**??**)

9.12 TComponentList

9.12.1 Description

`TComponentList` is a TObjectList (266) 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 (266), `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 (253), TObjectList (266), TClassList (219)

9.12.2 Method overview

Page	Method	Description
222	Add	Add a component to the list.
222	Destroy	Destroys the instance.
222	Extract	Remove a component from the list without destroying it.
223	First	First non-nil instance in the list.
223	IndexOf	Search for an instance in the list.
224	Insert	Insert a new component in the list.
224	Last	Last non-nil instance in the list.
223	Remove	Remove a component from the list, possibly destroying it.

9.12.3 Property overview

Page	Properties	Access	Description
224	Items	rw	Index-based access to the elements in the list.

9.12.4 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 (266), #rtl.classes.TComponent (**??**)

9.12.5 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 (267)

9.12.6 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 (266) 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 (223), TObjectList.Extract (267)

9.12.7 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 (266) so it only accepts `TComponent` descendents. It introduces no new behaviour.

Errors: None.

See also: TComponentList.Extract (222), TObjectList.Remove (268)

9.12.8 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 (268)

9.12.9 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 (224), TObjectList.First (269)

9.12.10 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 (223), TObjectList.Last (269)

9.12.11 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 (269), TComponentList.Add (222), TComponentList.Remove (223)

9.12.12 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 (270)

9.13 TCustomBucketList

9.13.1 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 (218). 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 (224).

See also: TBucketList (218)

9.13.2 Method overview

Page	Method	Description
226	Add	Add an item to the list.
226	Assign	Assign one bucket list to another.
225	Clear	Clear the list.
225	Destroy	Frees the bucketlist from memory.
226	Exists	Check if an item exists in the list.
226	Find	Find an item in the list.
227	ForEach	Loop over all items.
227	Remove	Remove an item from the list.

9.13.3 Property overview

Page	Properties	Access	Description
227	Data	rw	Associative array for data pointers.

9.13.4 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.

9.13.5 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 (226)

9.13.6 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 (226), TCustomBucketList.Clear (225)

9.13.7 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 (226), TCustomBucketList.Clear (225)

9.13.8 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 (226).

See also: TCustomBucketList.Find (226)

9.13.9 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 (226)

9.13.10 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 (227)

9.13.11 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 (226)

9.13.12 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 (226), TCustomBucketList.Exists (226)

9.14 TFPCustomHashTable

9.14.1 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 (217).

The TFPHashList (234) 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 (262), TFPObjectList (253), RSHash (217)

9.14.2 Method overview

Page	Method	Description
229	ChangeTableSize	Change the table size of the hash table.
229	Clear	Clear the hash table.
228	Create	Instantiate a new `TFPCustomHashTable` instance using the default hash mechanism.
229	CreateWith	Instantiate a new `TFPCustomHashTable` instance with given algorithm and size.
230	Delete	Delete a key from the hash list.
229	Destroy	Free the hash table.
230	Find	Search for an item with a certain key value.
230	IsEmpty	Check if the hash table is empty.

9.14.3 Property overview

Page	Properties	Access	Description
232	AVGChainLen	r	Average chain length.
231	Count	r	Number of items in the hash table.
233	Density	r	Number of filled slots.
230	HashFunction	rw	Hash function currently in use.
231	HashTable	r	Hash table instance.
231	HashTableSize	rw	Size of the hash table.
232	LoadFactor	r	Fraction of count versus size.
232	MaxChainLength	r	Maximum chain length.
232	NumberOfCollisions	r	Number of extra items.
231	VoidSlots	r	Number of empty slots in the hash table.

9.14.4 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 (217)

Errors: If no memory is available, an exception may be raised.

See also: CreateWith (229)

9.14.5 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 (228)

9.14.6 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 (229), Create (228), CreateWith (229)

9.14.7 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 (231)

9.14.8 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 (229)

9.14.9 TFPCustomHashTable.Delete

Synopsis: Delete a key from the hash list.

Declaration: `procedure Delete(const aKey: string);  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 (230)

9.14.10 TFPCustomHashTable.Find

Synopsis: Search for an item with a certain key value.

Declaration: `function Find(const aKey: string) : THTCustomNode`

Visibility: public

Description: `Find` searches for the THTCustomNode (262) 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 (230) method should be used instead.

Errors: None.

See also: Delete (230)

9.14.11 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 (231), TFPCustomHashTable.HashTableSize (231), TFPCustomHashTable.AVGChainLen (232), TFPCustomHashTable.MaxChainLength (232)

9.14.12 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 (231)

9.14.13 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 (230), TFPCustomHashTable.HashTableSize (231), TFPCustomHashTable.AVGChainLen (232), TFPCustomHashTable.MaxChainLength (232)

9.14.14 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 (230), TFPCustomHashTable.Count (231), TFPCustomHashTable.AVGChainLen (232), TFPCustomHashTable.MaxChainLength (232), TFPCustomHashTable.VoidSlots (231), TFPCustomHashTable.Density (233)

9.14.15 TFPCustomHashTable.HashTable

Synopsis: Hash table instance.

Declaration: `Property HashTable : TFPObjectList`

Visibility: public

Access: Read

Description: `TFPCustomHashTable` is the internal list object (TFPObjectList (253) used for the hash table. Each element in this table is again a TFPObjectList (253) instance or `Nil`.

9.14.16 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 (230), TFPCustomHashTable.Count (231), TFPCustomHashTable.AVGChainLen (232), TFPCustomHashTable.MaxChainLength (232), TFPCustomHashTable.LoadFactor (232), TFPCustomHashTable.Density (233), TFPCustomHashTable.NumberOfCollisions (232)

9.14.17 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 (230), TFPCustomHashTable.Count (231), TFPCustomHashTable.AVGChainLen (232), TFPCustomHashTable.MaxChainLength (232), TFPCustomHashTable.VoidSlots (231), TFPCustomHashTable.Density (233), TFPCustomHashTable.NumberOfCollisions (232)

9.14.18 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 (230), TFPCustomHashTable.Count (231), TFPCustomHashTable.LoadFactor (232), TFPCustomHashTable.MaxChainLength (232), TFPCustomHashTable.VoidSlots (231), TFPCustomHashTable.Density (233), TFPCustomHashTable.NumberOfCollisions (232)

9.14.19 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 (230), TFPCustomHashTable.Count (231), TFPCustomHashTable.LoadFactor (232), TFPCustomHashTable.AVGChainLen (232), TFPCustomHashTable.VoidSlots (231), TFPCustomHashTable.Density (233), TFPCustomHashTable.NumberOfCollisions (232)

9.14.20 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 (230), TFPCustomHashTable.Count (231), TFPCustomHashTable.LoadFactor (232), TFPCustomHashTable.AVGChainLen (232), TFPCustomHashTable.VoidSlots (231), TFPCustomHashTable.Density (233)

9.14.21 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 (230), TFPCustomHashTable.Count (231), TFPCustomHashTable.LoadFactor (232), TFPCustomHashTable.AVGChainLen (232), TFPCustomHashTable.VoidSlots (231), TFPCustomHashTable.Density (233)

9.15 TFPDataHashTable

9.15.1 Description

`TFPDataHashTable` is a TFPCustomHashTable (227) 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 (251), or for string data, TFPStringHashTable (260) is more suitable. The data pointers are exposed with their keys through the Items (234) property.

See also: TFPObjectHashTable (251), TFPStringHashTable (260), Items (234)

9.15.2 Method overview

Page	Method	Description
234	Add	Add a data pointer to the list.
233	Iterate	Iterate over the pointers in the hash table.

9.15.3 Property overview

Page	Properties	Access	Description
234	Items	rw	Key-based access to the items in the table.

9.15.4 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.

9.15.5 TFPDataHashTable.Add

Synopsis: Add a data pointer to the list.

Declaration: `procedure Add(const aKey: string; 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 (234)

9.15.6 TFPDataHashTable.Items

Synopsis: Key-based access to the items in the table.

Declaration: `Property Items[index: string]: 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 (261)

9.16 TFPHashList

9.16.1 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 (227) class can be used. If a hash class for objects is needed instead of pointers, the TFPHashObjectList (244) class can be used.

See also: TFPCustomHashTable (227), TFPHashObjectList (244), TFPDataHashTable (233), TFPStringHashTable (260)

9.16.2 Method overview

Page	Method	Description
236	Add	Add a new key/data pair to the list.
236	Clear	Clear the list.
235	Create	Create a new instance of the hashlist.
237	Delete	Delete an item from the list.
235	Destroy	Removes an instance of the hashlist from the heap.
237	Error	Raise an error.
237	Expand	Expand the list.
237	Extract	Extract a pointer from the list.
238	Find	Find data associated with key.
238	FindIndexOf	Return index of named item.
238	FindWithHash	Find first element with given name and hash value.
240	ForEachCall	Call a procedure for each element in the list.
237	GetNextCollision	Get next collision number.
236	HashOfIndex	Return the hash value of an item by index.
238	IndexOf	Return the index of the data pointer.
236	NameOfIndex	Returns the key name of an item by index.
239	Pack	Remove nil pointers from the list.
239	Remove	Remove first instance of a pointer.
239	Rename	Rename a key.
239	ShowStatistics	Return some statistics for the list.

9.16.3 Property overview

Page	Properties	Access	Description
240	Capacity	rw	Capacity of the list.
240	Count	rw	Current number of elements in the list.
240	Items	rw	Indexed array with pointers.
241	List	r	Low-level hash list.
241	Strs	r	Low-level memory area with strings.

9.16.4 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 (235)

9.16.5 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 (235), TFPHashList.Clear (236)

9.16.6 TFPHashList.Add

Synopsis: Add a new key/data pair to the list.

Declaration: `function Add(const AName: shortstring; Item: Pointer) : Integer`

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 (237), TFPHashList.Remove (239), TFPHashList.Delete (237)

9.16.7 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 (237), TFPHashList.Remove (239), TFPHashList.Delete (237), TFPHashList.Add (236)

9.16.8 TFPHashList.NameOfIndex

Synopsis: Returns the key name of an item by index.

Declaration: `function NameOfIndex(Index: Integer) : ShortString`

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 (236), TFPHashList.Find (238), TFPHashList.FindIndexOf (238), TFPHashList.FindWithHash (238)

9.16.9 TFPHashList.HashOfIndex

Synopsis: Return the hash value of an item by index.

Declaration: `function HashOfIndex(Index: Integer) : 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 (236), TFPHashList.Find (238), TFPHashList.FindIndexOf (238), TFPHashList.FindWithHash (238)

9.16.10 TFPHashList.GetNextCollision

Synopsis: Get next collision number.

Declaration: `function GetNextCollision(Index: Integer) : Integer`

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

9.16.11 TFPHashList.Delete

Synopsis: Delete an item from the list.

Declaration: `procedure Delete(Index: Integer)`

Visibility: public

Description: `Delete` deletes the item at position `Index`. The data to which it points is not freed from memory.

Errors: TFPHashList.Extract (237)TFPHashList.Remove (239)TFPHashList.Add (236)

9.16.12 TFPHashList.Error

Synopsis: Raise an error.

Declaration: `class procedure Error(const Msg: string; Data: PtrInt)`

Visibility: public

Description: `Error` raises an `EListError` exception, with message `Msg`. The `Data` pointer is used to format the message.

9.16.13 TFPHashList.Expand

Synopsis: Expand the list.

Declaration: `function Expand : TFPHashList`

Visibility: public

Description: `Expand` enlarges the capacity of the list if the maximum capacity was reached. It returns itself.

Errors: If not enough memory is available, an exception may be raised.

See also: TFPHashList.Clear (236)

9.16.14 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 (237), TFPHashList.Remove (239), TFPHashList.Clear (236)

9.16.15 TFPHashList.IndexOf

Synopsis: Return the index of the data pointer.

Declaration: `function IndexOf(Item: Pointer) : Integer`

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 (236), TFPHashList.NameOfIndex (236), TFPHashList.Find (238), TFPHashList.FindIndexOf (238), TFPHashList.FindWithHash (238)

9.16.16 TFPHashList.Find

Synopsis: Find data associated with key.

Declaration: `function Find(const AName: shortstring) : 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 (236), TFPHashList.NameOfIndex (236), TFPHashList.IndexOf (238), TFPHashList.FindIndexOf (238), TFPHashList.FindWithHash (238)

9.16.17 TFPHashList.FindIndexOf

Synopsis: Return index of named item.

Declaration: `function FindIndexOf(const AName: shortstring) : 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. Note that `Nil` data pointers will result in -1 as well.

See also: TFPHashList.HashOfIndex (236), TFPHashList.NameOfIndex (236), TFPHashList.IndexOf (238), TFPHashList.Find (238), TFPHashList.FindWithHash (238)

9.16.18 TFPHashList.FindWithHash

Synopsis: Find first element with given name and hash value.

Declaration:
```
function FindWithHash(const AName: shortstring; 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 (236), TFPHashList.NameOfIndex (236), TFPHashList.IndexOf (238), TFPHashList.Find (238), TFPHashList.FindIndexOf (238)

9.16.19 TFPHashList.Rename

Synopsis: Rename a key.

Declaration:
```
function Rename(const AOldName: shortstring;
                const ANewName: shortstring) : Integer
```

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.

9.16.20 TFPHashList.Remove

Synopsis: Remove first instance of a pointer.

Declaration:
```
function Remove(Item: Pointer) : Integer
```

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 (237), TFPHashList.Clear (236), TFPHashList.Extract (237)

9.16.21 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 (236)

9.16.22 TFPHashList.ShowStatistics

Synopsis: Return some statistics for the list.

Declaration:
```
procedure ShowStatistics
```

Visibility: public

Description: `ShowStatistics` prints some information about the hash list to standard output. It prints the following values:

HashSizeSize of the hash table

HashMeanMean hash value

HashStdDevStandard deviation of hash values

ListSizeSize and capacity of the list

StringSizeSize and capacity of key strings

9.16.23 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`.

9.16.24 TFPHashList.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 (240), Items (240)

9.16.25 TFPHashList.Count

Synopsis: Current number of elements in the list.

Declaration:
```
Property Count : Integer
```

Visibility: public

Access: Read,Write

Description: `Count` is the current number of elements in the list.

See also: Capacity (240), Items (240)

9.16.26 TFPHashList.Items

Synopsis: Indexed array with pointers.

Declaration:
```
Property Items[Index: Integer]: Pointer; default
```

Visibility: public

Access: Read,Write

Description: `Items` provides indexed access to the pointers, the index runs from 0 to Count-1 (240).

Errors: Specifying an invalid index will result in an exception.

See also: Capacity (240), Count (240)

9.16.27 TFPHashList.List

Synopsis: Low-level hash list.

Declaration: `Property List : PHashItemList`

Visibility: public

Access: Read

Description: `List` exposes the low-level item list (215). It should not be used directly.

See also: Strs (241), THashItemList (215)

9.16.28 TFPHashList.Strs

Synopsis: Low-level memory area with strings.

Declaration: `Property Strs : PChar`

Visibility: public

Access: Read

Description: `Strs` exposes the raw memory area with the strings.

See also: List (241)

9.17 TFPHashObject

9.17.1 Description

`TFPHashObject` is a `TObject` descendent which is aware of the TFPHashObjectList (244) 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 (243), TFPHashObject.ChangeOwner (242), TFPHashObject.ChangeOwnerAndName (242)

9.17.2 Method overview

Page	Method	Description
242	ChangeOwner	Change the list owning the object.
242	ChangeOwnerAndName	Simultaneously change the list owning the object and the name of the object.
242	Create	Create a named instance, and insert in a hash list.
242	CreateNotOwned	Create an instance not owned by any list.
243	Rename	Rename the object.

9.17.3 Property overview

Page	Properties	Access	Description
243	Hash	r	Hash value.
243	Name	r	Current name of the object.

9.17.4 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 (244) hash list. It also has no name when created in this way.

See also: TFPHashObject.Name (243), TFPHashObject.ChangeOwner (242), TFPHashObject.ChangeOwnerAndName (242)

9.17.5 TFPHashObject.Create

Synopsis: Create a named instance, and insert in a hash list.

Declaration:
```
constructor Create(HashObjectList: TFPHashObjectList;
                   const s: shortstring)
```

Visibility: public

Description: `Create` creates an instance of `TFPHashObject`, gives it the name `S` and inserts it in the hash list HashObjectList (244).

See also: CreateNotOwned (242), TFPHashObject.ChangeOwner (242), TFPHashObject.Name (243)

9.17.6 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 (242), Name (243)

9.17.7 TFPHashObject.ChangeOwnerAndName

Synopsis: Simultaneously change the list owning the object and the name of the object.

Declaration:
```
procedure ChangeOwnerAndName(HashObjectList: TFPHashObjectList;
                             const s: shortstring)
```

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 (242), Name (243)

9.17.8 TFPHashObject.Rename

Synopsis: Rename the object.

Declaration: `procedure Rename(const ANewName: shortstring)`

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 (242), ChangeOwnerAndName (242), Name (243)

9.17.9 TFPHashObject.Name

Synopsis: Current name of the object.

Declaration: `Property Name : shortstring`

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 (243), ChangeOwnerAndName (242)

9.17.10 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 (243)

9.18 TFPHashObjectList

9.18.1 Method overview

Page	Method	Description
245	Add	Add a new key/data pair to the list.
245	Clear	Clear the list.
244	Create	Create a new instance of the hashlist.
246	Delete	Delete an object from the list.
244	Destroy	Removes an instance of the hashlist from the heap.
246	Expand	Expand the list.
247	Extract	Extract a object instance from the list.
247	Find	Find data associated with key.
248	FindIndexOf	Return index of named object.
248	FindInstanceOf	Search an instance of a certain class.
248	FindWithHash	Find first element with given name and hash value.
249	ForEachCall	Call a procedure for each object in the list.
246	GetNextCollision	Get next collision number.
246	HashOfIndex	Return the hash value of an object by index.
247	IndexOf	Return the index of the object instance.
245	NameOfIndex	Returns the key name of an object by index.
249	Pack	Remove nil object instances from the list.
247	Remove	Remove first occurrence of a object instance.
248	Rename	Rename a key.
249	ShowStatistics	Return some statistics for the list.

9.18.2 Property overview

Page	Properties	Access	Description
249	Capacity	rw	Capacity of the list.
250	Count	rw	Current number of elements in the list.
250	Items	rw	Indexed array with object instances.
250	List	r	Low-level hash list.
250	OwnsObjects	rw	Does the list own the objects it contains.

9.18.3 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 (244), TFPHashObjectList.OwnsObjects (250)

9.18.4 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 (244), TFPHashObjectList.Clear (245)

9.18.5 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 (250) is `True`. It always frees all memory needed to contain the objects.

Errors: None.

See also: TFPHashObjectList.Extract (247), TFPHashObjectList.Remove (247), TFPHashObjectList.Delete (246), TFPHashObjectList.Add (245)

9.18.6 TFPHashObjectList.Add

Synopsis: Add a new key/data pair to the list.

Declaration: `function Add(const AName: shortstring; 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 (247), TFPHashObjectList.Remove (247), TFPHashObjectList.Delete (246)

9.18.7 TFPHashObjectList.NameOfIndex

Synopsis: Returns the key name of an object by index.

Declaration: `function NameOfIndex(Index: Integer) : ShortString`

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 (246), TFPHashObjectList.Find (247), TFPHashObjectList.FindIndexOf (248), TFPHashObjectList.FindWithHash (248)

9.18.8 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 (245), TFPHashObjectList.Find (247), TFPHashObjectList.FindIndexOf (248), TFPHashObjectList.FindWithHash (248)

9.18.9 TFPHashObjectList.GetNextCollision

Synopsis: Get next collision number.

Declaration: `function GetNextCollision(Index: Integer) : Integer`

Visibility: public

Description: Get next collision number.

9.18.10 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 (250) is `True`, then the object itself is also freed from memory.

See also: TFPHashObjectList.Extract (247), TFPHashObjectList.Remove (247), TFPHashObjectList.Add (245), OwnsObjects (250)

9.18.11 TFPHashObjectList.Expand

Synopsis: Expand the list.

Declaration: `function Expand : TFPHashObjectList`

Visibility: public

Description: `Expand` enlarges the capacity of the list if the maximum capacity was reached. It returns itself.

Errors: If not enough memory is available, an exception may be raised.

See also: TFPHashObjectList.Clear (245)

9.18.12 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 (250).

`Extract` does a linear search, and is not very efficient.

See also: TFPHashObjectList.Delete (246), TFPHashObjectList.Remove (247), TFPHashObjectList.Clear (245)

9.18.13 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 (250) is `True`, then the object itself is also freed from memory.

See also: TFPHashObjectList.Delete (246), TFPHashObjectList.Clear (245), TFPHashObjectList.Extract (247)

9.18.14 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 (246), TFPHashObjectList.NameOfIndex (245), TFPHashObjectList.Find (247), TFPHashObjectList.FindIndexOf (248), TFPHashObjectList.FindWithHash (248)

9.18.15 TFPHashObjectList.Find

Synopsis: Find data associated with key.

Declaration: `function Find(const s: shortstring) : 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 (246), TFPHashObjectList.NameOfIndex (245), TFPHashObjectList.IndexOf (247), TFPHashObjectList.FindIndexOf (248), TFPHashObjectList.FindWithHash (248)

9.18.16 TFPHashObjectList.FindIndexOf

Synopsis: Return index of named object.

Declaration:
```
function FindIndexOf(const s: shortstring) : 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 (246), TFPHashObjectList.NameOfIndex (245), TFPHashObjectList.IndexOf (247), TFPHashObjectList.Find (247), TFPHashObjectList.FindWithHash (248)

9.18.17 TFPHashObjectList.FindWithHash

Synopsis: Find first element with given name and hash value.

Declaration:
```
function FindWithHash(const AName: shortstring; 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 (246), TFPHashObjectList.NameOfIndex (245), TFPHashObjectList.IndexOf (247), TFPHashObjectList.Find (247), TFPHashObjectList.FindIndexOf (248)

9.18.18 TFPHashObjectList.Rename

Synopsis: Rename a key.

Declaration:
```
function Rename(const AOldName: shortstring;
                const ANewName: shortstring) : 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.

9.18.19 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.

9.18.20 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 (245)

9.18.21 TFPHashObjectList.ShowStatistics

Synopsis: Return some statistics for the list.

Declaration:
```
procedure ShowStatistics
```

Visibility: public

Description: `ShowStatistics` prints some information about the hash list to standard output. It prints the following values:

HashSizeSize of the hash table

HashMeanMean hash value

HashStdDevStandard deviation of hash values

ListSizeSize and capacity of the list

StringSizeSize and capacity of key strings

9.18.22 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`.

9.18.23 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 (250), Items (250)

9.18.24 TFPHashObjectList.Count

Synopsis: Current number of elements in the list.

Declaration: `Property Count : Integer`

Visibility: public

Access: Read,Write

Description: `Count` is the current number of elements in the list.

See also: Capacity (249), Items (250)

9.18.25 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 (244)

9.18.26 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 (250).

Errors: Specifying an invalid index will result in an exception.

See also: Capacity (249), Count (250)

9.18.27 TFPHashObjectList.List

Synopsis: Low-level hash list.

Declaration: `Property List : TFPHashList`

Visibility: public

Access: Read

Description: `List` exposes the low-level hash list (234). It should not be used directly.

See also: TFPHashList (234)

9.19 TFPObjectHashTable

9.19.1 Description

`TFPStringHashTable` is a TFPCustomHashTable (227) 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 (260), or for arbitrary pointer data, TFPDataHashTable (233) is more suitable. The objects are exposed with their keys through the Items (252) property.

See also: TFPStringHashTable (260), TFPDataHashTable (233), TFPObjectHashTable.Items (252)

9.19.2 Method overview

Page	Method	Description
252	Add	Add a new object to the hash table.
251	Create	Create a new instance of `TFPObjectHashTable`.
251	CreateWith	Create a new hash table with given size and hash function.
252	Iterate	Iterate over the objects in the hash table.

9.19.3 Property overview

Page	Properties	Access	Description
252	Items	rw	Key-based access to the objects.
252	OwnsObjects	r	Does the hash table own the objects ?

9.19.4 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 (252) 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 (252), TFPObjectHashTable.CreateWith (251), TFPObjectHashTable.Items (252)

9.19.5 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 (252) 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 (252), TFPObjectHashTable.Create (251), TFPObjectHashTable.Items (252)

9.19.6 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.

9.19.7 TFPObjectHashTable.Add

Synopsis: Add a new object to the hash table.

Declaration:
```
procedure Add(const aKey: string; 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 (252)

9.19.8 TFPObjectHashTable.Items

Synopsis: Key-based access to the objects.

Declaration:
```
Property Items[index: string]: 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 (252)

9.19.9 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 (251), TFPObjectHashTable.Items (252)

9.20 TFPObjectList

9.20.1 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 (266), `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 (266)

9.20.2 Method overview

Page	Method	Description
254	Add	Add an object to the list.
258	Assign	Copy the contents of a list.
254	Clear	Clear all elements in the list.
254	Create	Create a new object list.
255	Delete	Delete an element from the list.
254	Destroy	Clears the list and destroys the list instance.
255	Exchange	Exchange the location of two objects.
255	Expand	Expand the capacity of the list.
256	Extract	Extract an object from the list.
256	FindInstanceOf	Search for an instance of a certain class.
257	First	Return the first non-nil object in the list.
259	ForEachCall	For each object in the list, call a method or procedure, passing it the object.
256	IndexOf	Search for an object in the list.
257	Insert	Insert a new object in the list.
257	Last	Return the last non-nil object in the list.
258	Move	Move an object to another location in the list.
258	Pack	Remove all `Nil` references from the list.
256	Remove	Remove an item from the list.
258	Sort	Sort the list of objects.

9.20.3 Property overview

Page	Properties	Access	Description
259	Capacity	rw	Capacity of the list.
259	Count	rw	Number of elements in the list.
260	Items	rw	Indexed access to the elements of the list.
260	List	r	Internal list used to keep the objects.
260	OwnsObjects	rw	Should the list free elements when they are removed.

9.20.4 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 (254), TFPObjectList.OwnsObjects (260), TObjectList (266)

9.20.5 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 (260) is `True`.

See also: TFPObjectList.OwnsObjects (260), TObjectList.Create (267)

9.20.6 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 (260) is `True`.

9.20.7 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 (260) 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 (260), TFPObjectList.Delete (255)

9.20.8 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 (260) is `True`, the object is also freed.

Errors: An access violation may occur when OwnsObjects (260) is `True` and either the object was freed externally, or when the same object is in the same list twice.

See also: TFPObjectList.Remove (256), TFPObjectList.Extract (256), TFPObjectList.OwnsObjects (260), TFPObjectList.Add (254), TFPObjectList.Clear (254)

9.20.9 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 (254), TFPObjectList.Delete (255)

9.20.10 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 (258), TFPObjectList.Clear (254), #rtl.classes.tfplist.expand (**??**)

9.20.11 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 (258), TFPObjectList.Clear (254), TFPObjectList.Remove (256), TFPObjectList.Delete (255)

9.20.12 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 (260) 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 (258), TFPObjectList.Clear (254), TFPObjectList.Delete (255), TFPObjectList.Extract (256)

9.20.13 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 (260), TFPObjectList.Remove (256), TFPObjectList.Extract (256)

9.20.14 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 (256)

9.20.15 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 (254), TFPObjectList.Delete (255)

9.20.16 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 (257), TFPObjectList.Pack (258)

9.20.17 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 (257), TFPObjectList.Pack (258)

9.20.18 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 (255), 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 (255), TFPObjectList.Delete (255), TFPObjectList.Insert (257)

9.20.19 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.

9.20.20 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 (257), TFPObjectList.Last (257)

9.20.21 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 (**??**)

9.20.22 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 (216), TObjectListCallback (216)

9.20.23 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 (259).

See also: TFPObjectList.Count (259)

9.20.24 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 (259)

9.20.25 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 (254), TFPObjectList.Delete (255), TFPObjectList.Remove (256), TFPObjectList.Clear (254)

9.20.26 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 (259)

9.20.27 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 (**??**)

9.21 TFPStringHashTable

9.21.1 Description

`TFPStringHashTable` is a TFPCustomHashTable (227) 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 (251), or for arbitrary pointer data, TFPDataHashTable (233) is more suitable. The strings are exposed with their keys through the Items (261) property.

See also: TFPObjectHashTable (251), TFPDataHashTable (233), Items (261)

9.21.2 Method overview

Page	Method	Description
261	Add	Add a new string to the hash list.
261	Iterate	Iterate over the strings in the hash table.

9.21.3 Property overview

Page	Properties	Access	Description
261	Items	rw	Key based access to the strings in the hash table.

9.21.4 TFPStringHashTable.Iterate

Synopsis: Iterate over the strings in the hash table.

Declaration:
```
function Iterate(aMethod: TStringIteratorMethod) : string;  Virtual
function Iterate(aMethod: TStringIteratorCallback) : string;  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.

9.21.5 TFPStringHashTable.Add

Synopsis: Add a new string to the hash list.

Declaration:
```
procedure Add(const aKey: string; const aItem: string);  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 (261)

9.21.6 TFPStringHashTable.Items

Synopsis: Key based access to the strings in the hash table.

Declaration:
```
Property Items[index: string]: string; 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 (261)

9.22 THTCustomNode

9.22.1 Description

`THTCustomNode` is used by the TFPCustomHashTable (227) class to store the keys and associated values.

See also: TFPCustomHashTable (227)

9.22.2 Method overview

Page	Method	Description
262	CreateWith	Create a new instance of `THTCustomNode`.
262	HasKey	Check whether this node matches the given key.

9.22.3 Property overview

Page	Properties	Access	Description
263	Key	r	Key value associated with this hash item.

9.22.4 THTCustomNode.CreateWith

Synopsis: Create a new instance of `THTCustomNode`.

Declaration: `constructor CreateWith(const AString: string)`

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 (227) class when needed.

Errors: If no more memory is available, an exception may be raised.

See also: TFPCustomHashTable (227)

9.22.5 THTCustomNode.HasKey

Synopsis: Check whether this node matches the given key.

Declaration: `function HasKey(const AKey: string) : 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 (263)

9.22.6 THTCustomNode.Key

Synopsis: Key value associated with this hash item.

Declaration: `Property Key : string`

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 (262)

9.23 THTDataNode

9.23.1 Description

`THTDataNode` is used by TFPDataHashTable (233) 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 (233), THTObjectNode (263), THTStringNode (264)

9.23.2 Property overview

Page	Properties	Access	Description
263	Data	rw	Data pointer.

9.23.3 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.

9.24 THTObjectNode

9.24.1 Description

`THTObjectNode` is a THTCustomNode (262) descendent which holds the data in the TFPObjectHashTable (251) 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 (251)

9.24.2 Property overview

Page	Properties	Access	Description
264	Data	rw	Object instance.

9.24.3 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 (252)

See also: TFPObjectHashTable (251), TFPObjectHashTable.Items (252), THTOwnedObjectNode (264)

9.25 THTOwnedObjectNode

9.25.1 Description

`THTOwnedObjectNode` is used instead of THTObjectNode (263) in case TFPObjectHashTable (251) owns it's objects. When this object is destroyed, the associated data object is also destroyed.

See also: TFPObjectHashTable (251), THTObjectNode (263), TFPObjectHashTable.OwnsObjects (252)

9.25.2 Method overview

Page	Method	Description
264	Destroy	Destroys the node and the object.

9.25.3 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 (264), TFPObjectHashTable.OwnsObjects (252)

9.26 THTStringNode

9.26.1 Description

`THTStringNode` is a THTCustomNode (262) descendent which holds the data in the TFPString-HashTable (260) 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 (260)

9.26.2 Property overview

Page	Properties	Access	Description
265	Data	rw	String data.

9.26.3 THTStringNode.Data

Synopsis: String data.

Declaration: `Property Data : string`

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 (261)

See also: TFPStringHashTable (260)

9.27 TObjectBucketList

9.27.1 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 (265)

9.27.2 Method overview

Page	Method	Description
265	Add	Add an object to the list.
266	Remove	Remove an object from the list.

9.27.3 Property overview

Page	Properties	Access	Description
266	Data	rw	Associative array of data items.

9.27.4 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 (266), TObjectBucketList.Remove (266)

9.27.5 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 (265), TObjectBucketList.Data (266)

9.27.6 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 (265)

9.28 TObjectList

9.28.1 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 (253), `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 (253), TComponentList (221), TClassList (219)

9.28.2 Method overview

Page	Method	Description
267	Add	Add an object to the list.
267	Create	Create a new object list.
267	Extract	Extract an object from the list.
268	FindInstanceOf	Search for an instance of a certain class.
269	First	Return the first non-nil object in the list.
268	IndexOf	Search for an object in the list.
269	Insert	Insert an object in the list.
269	Last	Return the last non-nil object in the list.
268	Remove	Remove (and possibly free) an element from the list.

9.28.3 Property overview

Page	Properties	Access	Description
270	Items	rw	Indexed access to the elements of the list.
269	OwnsObjects	rw	Should the list free elements when they are removed.

9.28.4 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 (269), TFPObjectList (253)

9.28.5 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 (269), #rtl.classes.TList.Delete (**??**), TObjectList.Extract (267), TObjectList.Remove (268)

9.28.6 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 (268), `Extract` does not free the extracted element if OwnsObjects (269) 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 (268)

9.28.7 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 (269) 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 (267)

9.28.8 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 (268)

9.28.9 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 (268)

9.28.10 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 (267), TObjectList.Remove (268)

9.28.11 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 (269)

9.28.12 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 (269)

9.28.13 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 (267), TObjectList.Remove (268), TObjectList.Extract (267)

9.28.14 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 (**??**)

9.29 TObjectQueue

9.29.1 Method overview

Page	Method	Description
271	Peek	Look at the first object in the queue.
270	Pop	Pop the first element off the queue.
270	Push	Push an object on the queue.

9.29.2 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 (270), TObjectQueue.Peek (271)

9.29.3 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 (270), TObjectQueue.Peek (271)

9.29.4 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 (270), TObjectQueue.Pop (270)

9.30 TObjectStack

9.30.1 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 (272), TStack (274), TQueue (274), TObjectQueue (270)

9.30.2 Method overview

Page	Method	Description
272	Peek	Look at the top object in the stack.
271	Pop	Pop the top object of the stack.
271	Push	Push an object on the stack.

9.30.3 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 (271), TObjectStack.Peek (272)

9.30.4 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 (271), TObjectStack.Peek (272)

9.30.5 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 (271), TObjectStack.Pop (271)

9.31 TOrderedList

9.31.1 Description

`TOrderedList` provides the base class for TQueue (274) and TStack (274). 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 (274), TStack (274)

9.31.2 Method overview

Page	Method	Description
273	AtLeast	Check whether the list contains a certain number of elements.
273	Count	Number of elements on the list.
272	Create	Create a new ordered list.
273	Destroy	Free an ordered list.
274	Peek	Return the next element to be popped from the list.
274	Pop	Remove an element from the list.
273	Push	Push another element on the list.

9.31.3 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 (273)

9.31.4 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 (272)

9.31.5 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 (273)

9.31.6 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 (273)

9.31.7 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 (274), TOrderedList.Peek (274)

9.31.8 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 (274), TOrderedList.Push (273)

9.31.9 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 (274), without actually popping it from the list.

Errors: None.

See also: TOrderedList.Pop (274), TOrderedList.Push (273)

9.32 TQueue

9.32.1 Description

`TQueue` is a descendent of TOrderedList (272) which implements Push (273) and Pop (274) 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 (272)

See also: TOrderedList (272), TObjectQueue (270), TStack (274)

9.33 TStack

9.33.1 Description

`TStack` is a descendent of TOrderedList (272) which implements Push (273) and Pop (274) 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 (272)

See also: TOrderedList (272), TObjectStack (271), TQueue (274)

Chapter 10

Reference for unit 'csvdocument'

10.1 Used units

Table 10.1: Used units by unit 'csvdocument'

Name	Page
bufstream	197
Classes	??
Contnrs	213
csvreadwrite	284
System	??
sysutils	??

10.2 Overview

The `CSVDocument` unit offers the TCSVDocument (276) 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 (290) and #fcl.csvreadwrite.TCSVBuilder (285) units to read and write the actual file.

10.3 Constants, types and variables

10.3.1 Types

`TCSVBuilder = csvreadwrite.TCSVBuilder`

`TCSVBuilder` is a backwards-compatibility alias for csvreadwrite.TCSVParser (290)

`TCSVChar = csvreadwrite.TCSVChar`

`TCSVChar` is a backwards-compatibility alias for csvreadwrite.TCSVChar (285)

`TCSVParser = csvreadwrite.TCSVParser`

`TCSVParser` is a backwards-compatibility alias for csvreadwrite.TCSVParser (290)

10.4 TCSVDocument

10.4.1 Description

`TCSVDocument` can be used to read a CSV file in memory using e.g. LoadFromFile (277), manipulate the contents using the Cells (282) property. Additional rows can be added using AddRow (278), additional cells can be added using AddCell (278). After all is done, the SaveToFile (278) method can be used to save the new content to file. The various properties introduced in csvreadwrite.TCSVHandler (275) can be used to configure the format of the CSV file.

See also: LoadFromFile (277), Cells (282), AddRow (278), AddCell (278), SaveToFile (278), csvreadwrite.TCSVHandler (275)

10.4.2 Method overview

Page	Method	Description
278	AddCell	Add a new cell to a row.
278	AddRow	Add a new row to the CSV Data.
281	Clear	Remove all rows.
281	CloneRow	Duplicate a row.
276	Create	Create a new instance of `TCSVDocument`.
277	Destroy	Remove the `TCSVDocument` instance from memory.
281	ExchangeRows	Exchange positions of 2 rows.
280	HasCell	Test if a cell exists.
280	HasRow	Test if a row exists.
280	IndexOfCol	Test whether a value exists at a given row.
280	IndexOfRow	Test whether a value exists at a given column.
279	InsertCell	Insert cell at specified position.
279	InsertRow	Insert row before row number `aRow`.
277	LoadFromFile	Load a CSV file into the document.
277	LoadFromStream	Load CSV data from a stream.
279	RemoveCell	Remove a cell at a particular row.
279	RemoveRow	Remove a row.
281	RemoveTrailingEmptyCells	Remove empty cells at the end of each row.
278	SaveToFile	Save to file on disk.
278	SaveToStream	Save the CSV data to stream.
281	UnifyEmbeddedLineEndings	Ensures all values have the same line ending/.

10.4.3 Property overview

Page	Properties	Access	Description
282	Cells	rw	Array access to all cells.
282	ColCount	r	Return the column count for a given row.
283	CSVText	rw	Return the CSV Document as a single CSV text.
282	MaxColCount	r	Maximum column count.
282	RowCount	r	Number of available rows.

10.4.4 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 (282), Destroy (277)

10.4.5 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 (282), Create (276)

10.4.6 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 (277) 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 (277), SaveToStream (278), SaveToFile (278)

10.4.7 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 (288) when determining fields and rows. If EqualColCountPerRow (290) 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 (290), SaveToStream (278), SaveToFile (278), LoadFromFile (277), TCSVHandler (288)

10.4.8 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 (278) 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 (278), LoadFromStream (277), LoadFromFile (277)

10.4.9 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 (288) to apply the correct formatting to fields and rows. If EqualColCountPerRow (290) 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 (290), SaveToFile (278), LoadFromStream (277), LoadFromFile (277), TCSVHandler (288)

10.4.10 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 (278), InsertRow (279), InsertCell (279), RemoveRow (279)

10.4.11 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 (278), InsertRow (279), InsertCell (279), RemoveCell (279)

10.4.12 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 (278), AddCell (278), InsertCell (279), RemoveRow (279)

10.4.13 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 (278), InsertRow (279), AddCell (278), RemoveCell (279)

10.4.14 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 (278), RemoveCell (279)

10.4.15 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 (278), RemoveRow (279), HasCell (280)

10.4.16 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 (282).

See also: RowCount (282), HasCell (280)

10.4.17 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 (282)) 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] (282)) .

See also: RowCount (282), HasRow (280), ColCount (282)

10.4.18 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 (282), IndexOfRow (280)

10.4.19 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 (280), Cells (282)

10.4.20 TCSVDocument.Clear

Synopsis: Remove all rows.

Declaration: `procedure Clear`

Visibility: public

Description: `Clear` removes all rows from the document.

See also: AddRow (278)

10.4.21 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 (278), InsertRow (279), ExchangeRows (281)

10.4.22 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 (278), InsertRow (279), CloneRow (281)

10.4.23 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 (275) setting. Cell values that do not have a line-endings in them are left untouched.

See also: LineEnding (275), ChangeLineEndings (275)

10.4.24 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 (279)

10.4.25 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 (278), AddRow (278), ColCount (282), RowCount (282)

10.4.26 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 (282), Cells (282)

10.4.27 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 (282), Cells (282)

10.4.28 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 (282) value.

See also: RowCount (282), Cells (282), ColCount (282)

10.4.29 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 (278) and returns the resulting string.

See also: SaveToFile (278), SaveToStream (278)

Chapter 11

Reference for unit 'csvreadwrite'

11.1 Used units

Table 11.1: Used units by unit 'csvreadwrite'

Name	Page
Classes	??
strutils	??
System	??
sysutils	??

11.2 Overview

The csvreadwrite unit contains a class TCSVParser (290) which allows you to read a CSV file, and TCSVBuilder (285) 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 (276) class (a class that offers a matrix-like representation of the CVS data) to read and write the CSV file.

11.3 Constants, types and variables

11.3.1 Types

```
TCSVByteOrderMark = (bomNone,bomUTF8,bomUTF16LE,bomUTF16BE)
```

Table 11.2: 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 (294) 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 (285) and it's descendents to define the character type read from file.

11.4 Procedures and functions

11.4.1 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.

11.5 TCSVBuilder

11.5.1 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 (286), call AppendCell (287) for each cell you wish to write, and call AppendRow (287) when you want to start a new line. If you didn't specify an output stream, the output is available in DefaultOutput (287) or DefaultOutputAsString (287).

See also: SetOutput (286), AppendCell (287), AppendRow (287), DefaultOutput (287), DefaultOutputAsString (287)

11.5.2 Method overview

Page	Method	Description
287	AppendCell	Append a cell to the output.
287	AppendRow	Append a new row.
286	Create	Create a new instance of TCSVBuilder.
286	Destroy	Free a TCSVBuilder instance.
286	ResetBuilder	Reset values.
286	SetOutput	Set the output stream to write to.

11.5.3 Property overview

Page	Properties	Access	Description
287	DefaultOutput	r	Default output as stream.
287	DefaultOutputAsString	r	Default output as string value.

11.5.4 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 (287), DefaultOutputAsString (287)

11.5.5 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 (287), DefaultOutputAsString (287)

11.5.6 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 (287) or DefaultOutputAsString (287). Calling `SetOutpu` will result in resetting the inner state of the builder using ResetBuilder (286) , so it must not be called if you already wrote some data.

See also: DefaultOutput (287), DefaultOutputAsString (287), ResetBuilder (286)

11.5.7 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 (286) is called.

See also: SetOutput (286)

11.5.8 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 (289) if necessary, and append the necessary Delimiter (289) characters. It will not move to the next line, for this AppendRow (287) must be called.

See also: QuoteChar (289), Delimiter (289), AppendRow (287)

11.5.9 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 (289) character.

See also: LineENding (289), AppendCell (287), EqualColCountPerRow (290)

11.5.10 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 (286) call.

See also: SetOutput (286), DefaultOutputAsString (287)

11.5.11 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 (286) call. It is basically the contents of the DefaultOutput (287) property as a string.

See also: SetOutput (286), DefaultOutput (287)

11.6 TCSVHandler

11.6.1 Description

`TCSVHandler` is the base class for class TCSVParser (290) which allows you to read a CSV file, and class TCSVBuilder (285) 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 (289) or QuoteChar (289) and LineEnding (289) properties.

Normally you will not create an instance of `TCSVHandler`, instead you will create a `TCSVParser` or `TCSVBuilder` instance.

See also: TCSVParser (290), TCSVBuilder (285), Delimiter (289), QuoteChar (289), LineEnding (289)

11.6.2 Method overview

Page	Method	Description
288	Assign	Assign properties from another `TCSVHandler` instance.
289	AssignCSVProperties	Assign properties from another `TCSVHandler` instance.
288	Create	Create an instance of `TCSVHandler`.

11.6.3 Property overview

Page	Properties	Access	Description
289	Delimiter	rw	Field delimiter character.
290	EqualColCountPerRow	rw	Ensure every row has an equal amount of columns.
290	IgnoreOuterWhitespace	rw	Ignore whitespace between delimiters and field data.
289	LineEnding	rw	Line ending character.
289	QuoteChar	rw	Character to quote values.
290	QuoteOuterWhitespace	rw	Write quotes when outer whitespace is found in a value.

11.6.4 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 (289), QuoteChar (289) and LineEnding (289) to their initial values.

See also: Delimiter (289), QuoteChar (289), LineEnding (289)

11.6.5 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 (289) to do the actual copying.

See also: TPersistent.Assign (**??**)

11.6.6 TCSVHandler.AssignCSVProperties

Synopsis: Assign properties from another `TCSVHandler` instance.

Declaration: `procedure AssignCSVProperties(ASource: TCSVHandler)`

Visibility: public

Description: `AssignCSVProperties` is called by Assign (288) to copy all `TCSVHandler` properties from the `aSource` instance to the current instance.

See also: Assign (288)

11.6.7 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 (289).

See also: QuoteChar (289), LineEnding (289)

11.6.8 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 (289)) or line ending (as set in LineEnding (289)) characters lose their special meaning and are considered part of the value. The default quote character is the double quote (`"`).

See also: Delimiter (289), LineEnding (289)

11.6.9 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.

11.6.10 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 (289) when reading data. The default is `False`

See also: Delimiter (289), QuoteOuterWhitespace (290)

11.6.11 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 (289), which can alter the value when reading data depending on the setting of IgnoreOuterWhitespace (290). The default is `True`.

See also: Delimiter (289), IgnoreOuterWhitespace (290)

11.6.12 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 (290) or TCSVBuilder (285) classes. It is used in the TCSVDocument (284) class. The default is `True`.

See also: TCSVParser (290), TCSVBuilder (285), TCSVDocument (284)

11.7 TCSVParser

11.7.1 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 (292), after which ParseNextCell (292) can be called till it returns

false. The current cell value after `ParseNextCell` returns `True` is available in CurrentCellText (293). By tracking CurrentRow (292), CurrentCol (293) and MaxColCount (293), the structure of the CSV file can be determined.

See also: SetSource (292), ParseNextCell (292), CurrentCellText (293), CurrentRow (292), CurrentCol (293), MaxColCount (293)

11.7.2 Method overview

Page	Method	Description
291	Create	Create a new instance of `TCSVParser`.
291	Destroy	Free the `TCSVParser` instance.
292	ParseNextCell	Parse the next cell.
292	ResetParser	Reset the parser to its initial state.
292	SetSource	Set the CSV source data.

11.7.3 Property overview

Page	Properties	Access	Description
294	BOM	r	type of BOM marker found at the start of the data.
293	CurrentCellText	r	Current field value.
293	CurrentCol	r	Column (zero based) of the current field.
292	CurrentRow	r	Row (zero based) of the current field.
294	DetectBOM	rw	Must the parser attempt to read the BOM marker ?
294	FreeStream	rw	Does the parser free the stream when done ?
293	MaxColCount	r	Return the maximum column count encountered till now.

11.7.4 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 (288)

11.7.5 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 (291)

11.7.6 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 (292) 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 (294) 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 (294), ResetParser (292)

11.7.7 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 (292), CurrentCol (293) and MaxColCount (293) are all set to zero, and the output is cleared. The stream is put at position zero and if DetectBOM (294) is `True`, the BOM (294) marker is read.

See also: SetSource (292), CurrentRow (292), CurrentCol (293), MaxColCount (293), BOM (294)

11.7.8 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 (289), QuoteChar (289) and LineEnding (289) properties to determine the field boundaries. When it has returned `True`, the properties CurrentRow (292), CurrentCol (293), MaxColCount (293) and CurrentCellText (293) can be used to determine what field was read and what the contents of the field were.

See also: CurrentRow (292), CurrentCol (293), MaxColCount (293), CurrentCellText (293), Delimiter (289), QuoteChar (289), LineEnding (289)

11.7.9 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 (292), CurrentCol (293), MaxColCount (293), CurrentCellText (293)

11.7.10 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 (292), ParseNextCell (292), MaxColCount (293), CurrentCellText (293)

11.7.11 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 (290) property.

See also: CurrentRow (292), CurrentCol (293), MaxColCount (293), ParseNextCell (292), QuoteOuterWhitespace (290)

11.7.12 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 (292) is called, and consequently the final value is only available after `ParseNextValue` returned `False`.

See also: CurrentRow (292), CurrentCol (293), CurrentCellText (293), ParseNextCell (292)

11.7.13 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 (292)

See also: SetSource (292)

11.7.14 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 (292) and DetectBOM (294) is `True`. For a list of possible values, see TCSVByteOrderMark (284).

See also: TCSVByteOrderMark (284), TCSVParser.SetSource (292)

11.7.15 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 (292) will attempt to read the BOM marker when the CSV data is set using SetSource (292) The result of the detection is available the in BOM (294) property) after the source data is set. It follows that `DetectBOM` must be set before calling SetSource (292).

See also: SetSource (292), ResetParser (292), BOM (294)

Chapter 12

Reference for unit 'CustApp'

12.1 Used units

Table 12.1: Used units by unit 'CustApp'

Name	Page
Classes	??
singleinstance	838
System	??
sysutils	??

12.2 Overview

The CustApp unit implements the TCustomApplication (296) 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.

12.3 Constants, types and variables

12.3.1 Types

```
TEventLogTypes = Set of TEventType
```

TEventLogTypes is a set of TEventType (**??**), used in TCustomApplication.EventLogFilter (307) 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 (301) call.

12.3.2 Variables

```
CustomApplication : TCustomApplication = Nil
```

CustomApplication contains the global application instance. All descendents of TCustomApplication (296) 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.

12.4 TCustomApplication

12.4.1 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 (298), ShowException (299), OnException (304) and StopOnException (307).
- Command-line parameter parsing in FindOptionIndex (300), GetOptionValue (300), CheckOptions (301) and HasOption (301)
- Environment variable handling in GetEnvironmentList (303) and EnvironmentVariable (306).

Descendent classes need to override the DoRun protected method to implement the functionality of the program.

12.4.2 Method overview

Page	Method	Description
301	CheckOptions	Check whether all given options on the command-line are valid.
297	Create	Create a new instance of the `TCustomApplication` class.
298	Destroy	Destroys the `TCustomApplication` instance.
300	FindOptionIndex	Return the index of an option.
303	GetEnvironmentList	Return a list of environment variables.
302	GetNonOptions	Get all non-switch options.
300	GetOptionValue	Return the value of a command-line option.
301	GetOptionValues	Get the values for an option that may be specified multiple times.
298	HandleException	Handle an exception.
301	HasOption	Check whether an option was specified.
298	Initialize	Initialize the application.
303	Log	Write a message to the event log.
299	Run	Runs the application.
299	ShowException	Show an exception to the user.
299	Terminate	Terminate the application.

12.4.3 Property overview

Page	Properties	Access	Description
306	CaseSensitiveOptions	rw	Are options interpreted case sensitive or not.
305	ConsoleApplication	r	Is the application a console application or not.
306	EnvironmentVariable	r	Environment variable access.
307	EventLogFilter	rw	Event to filter events, before they are sent to the system log.
307	ExceptionExitCode	rw	ExitCode to use then terminating the program due to an exception.
303	ExeName	r	Name of the executable.
304	HelpFile	rw	Location of the application help file.
305	Location	r	Application location.
304	OnException	rw	Exception handling event.
306	OptionChar	rw	Command-line switch character.
306	ParamCount	r	Number of command-line parameters.
305	Params	r	Command-line parameters.
308	SingleInstance	r	Single instance used to control single application instance behaviour.
308	SingleInstanceClass	rw	Class to use when creating single instance.
308	SingleInstanceEnabled	rw	Enable single application instance control.
307	StopOnException	rw	Should the program loop stop on an exception.
304	Terminated	r	Was `Terminate` called or not.
304	Title	rw	Application title.

12.4.4 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 (298)

12.4.5 TCustomApplication.Destroy

Synopsis: Destroys the `TCustomApplication` instance.

Declaration: `destructor Destroy; Override`

Visibility: public

Description: `Destroy` simply calls the inherited `Destroy`.

See also: TCustomApplication.Create (297)

12.4.6 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 (304) 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 (299) 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 (307) property is set to `True`, the Terminated (304) property is set to `True`, which will cause the Run (299) loop to stop, and the application will exit.

See also: ShowException (299), StopOnException (307), Terminated (304), Run (299)

12.4.7 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 (299), TCustomApplication.Terminated (304)

12.4.8 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 (298). If TCustomApplication.StopOnException (307) 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 (298), TCustomApplication.StopOnException (307)

12.4.9 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 (298), TCustomApplication.StopOnException (307)

12.4.10 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 (304) property and properly shut down the application if it is set to `True`.

When `AExitCode` is specified, it will passed to System.ExitCode (295), 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 (307) will be used as the value for this argument.

See also: TCustomApplication.Terminated (304), TCustomApplication.Run (299), ExceptionExitCode (307), System.ExitCode (295)

12.4.11 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 (306) property, the search is performed case sensitive or case insensitive.

Options are identified as command-line parameters which start with OptionChar (306) (by default the dash ('-') character).

See also: HasOption (301), GetOptionValue (300), CheckOptions (301), CaseSensitiveOptions (306), OptionChar (306)

12.4.12 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 (306) property, the search is performed case sensitive or case insensitive.

Options are identified as command-line parameters which start with OptionChar (306) (by default the dash ('-') character).

If an option can appear multiple times, use TCustomApplication.GetOptionValues (301) to retrieve all values. This function only returns the value of the first occurrence of an option.

See also: FindOptionIndex (300), HasOption (301), CheckOptions (301), CaseSensitiveOptions (306), OptionChar (306), TCustomApplication.GetOptionValues (301)

12.4.13 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 (300) will only return the first occurrence of a value.

Errors: None.

See also: TCustomApplication.GetOptionValue (300)

12.4.14 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 (306) property, the search is performed case sensitive or case insensitive.

Options are identified as command-line parameters which start with OptionChar (306) (by default the dash ('-') character).

See also: FindOptionIndex (300), GetOptionValue (300), CheckOptions (301), CaseSensitiveOptions (306), OptionChar (306)

12.4.15 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 (306) (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 (300), GetOptionValue (300), HasOption (301), CaseSensitiveOptions (306), OptionChar (306)

12.4.16 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 (301). 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 (301), TCustomApplication.Checkoptions (301), TCustomApplication.GetOptionValue (300), TCustomApplication.GetOptionValues (301)

12.4.17 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 (306)

12.4.18 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 (573)

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 (**??**)

12.4.19 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 (**??**)

12.4.20 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 (304)

12.4.21 TCustomApplication.Terminated

Synopsis: Was `Terminate` called or not.

Declaration: `Property Terminated : Boolean`

Visibility: public

Access: Read

Description: `Terminated` indicates whether Terminate (299) 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 (299)

12.4.22 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 (304)

12.4.23 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 (299).

If the event is set, then it is called by the HandleException (298) routine. Do not use the `OnException` event directly, instead call `HandleException`.

See also: ShowException (299)

12.4.24 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 (**??**)

12.4.25 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 (305)

12.4.26 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 (306). It is equivalent to calling ParamStr (**??**).

See also: ParamCount (306), Paramstr (**??**)

12.4.27 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 (305) property.

See also: Params (305), Paramstr (**??**), ParamCount (**??**)

12.4.28 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 (303) can be used.

See also: GetEnvironmentList (303), TCustomApplication.Params (305)

12.4.29 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 (300), GetOptionValue (300), HasOption (301), CaseSensitiveOptions (306), CheckOptions (301)

12.4.30 TCustomApplication.CaseSensitiveOptions

Synopsis: Are options interpreted case sensitive or not.

Declaration: `Property CaseSensitiveOptions : Boolean`

Visibility: public

Access: Read,Write

Description: `CaseSensitiveOptions` determines whether FindOptionIndex (300) and CheckOptions (301) 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 (300), GetOptionValue (300), HasOption (301), OptionChar (306), CheckOptions (301)

12.4.31 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 (299) and HandleException (298) procedures in case of an unhandled exception in the `DoRun` code. If `StopOnException` is `True` then Terminate (299) will be called after the exception was handled.

See also: Run (299), HandleException (298), Terminate (299)

12.4.32 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 (299)

12.4.33 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 (303) 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 (303)

12.4.34 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 (308) 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 (308), TCustomApplication.SingleInstanceEnabled (308)

12.4.35 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 (308). The default class is determined by the global singleinstance default class as specified in #fcl.singleinstance.DefaultSingleInstanceCla (839).

See also: TCustomApplication.SingleInstance (308), DefaultSingleInstanceClass (839)

12.4.36 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 (308) using TCustomApplication.SingleInstanceClass (308) and starts the check to wee whether this application is a client or server instance.

See also: TCustomApplication.SingleInstance (308), TCustomApplication.SingleInstanceClass (308)

Chapter 13

Reference for unit 'daemonapp'

13.1 Used units

Table 13.1: Used units by unit 'daemonapp'

Name	Page
Classes	??
CustApp	295
eventlog	571
rtlconsts	??
System	??
sysutils	??

13.2 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 (327).

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 (331).

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 (334).

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 (573) 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.

13.3 Constants, types and variables

13.3.1 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.

13.3.2 Types

```
TCurrentStatus = (csStopped,csStartPending,csStopPending,csRunning
  ,
                  csContinuePending,csPausePending,csPaused)
```

Table 13.2: 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 (334) definition.

```
TCustomDaemonMapperClass = Class of TCustomDaemonMapper
```

`TCustomDaemonMapperClass` is the class of `TCustomDaemonMapper`. It is used in the RegisterDaemonMapper (316) 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 (317) 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 13.3: 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 13.4: 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 13.5: 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 13.6: 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 13.7: 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 13.8: Enumeration values for type TWinControlCode

Value	Explanation
wccHardwareProfileChange	
wccNetBindChange	
wccParamChange	
wccPowerEvent	
wccPreShutdown	
wccSessionChange	
wccShutdown	
wccTimeChange	
wccTriggerEvent	
wccUserModeReboot	

```
TWinControlCodes = Set of TWinControlCode
```

13.3.3 Variables

```
AppClass : TCustomDaemonApplicationClass
```

AppClass can be set to the class of a TCustomDaemonApplication (319) descendant. When the Application (315) 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 (334)) is created.

```
SStatus : Array[1..5] of string = ('Stop', 'Pause', 'Continue', 'Interrogate'
  , 'Shutdown')
```

Status message.

13.4 Procedures and functions

13.4.1 Application

Synopsis: Application instance.

Declaration: `function Application : TCustomDaemonApplication`

Visibility: default

Description: Application is the TCustomDaemonApplication (319) instance used by this application. The instance is created at the first invocation of this function, so it is possible to use RegisterDaemonApplicationClass (316) to register an alternative TCustomDaemonApplication class to run the application.

See also: TCustomDaemonApplication (319), RegisterDaemonApplicationClass (316)

13.4.2 DaemonError

Synopsis: Raise an EDaemon exception.

Declaration:
```
procedure DaemonError(Msg: string)
procedure DaemonError(Fmt: string; Args: Array of const)
```

Visibility: default

Description: DaemonError raises an EDaemon (316) exception with message Msg or it formats the message using Fmt and Args.

See also: EDaemon (316)

13.4.3 RegisterDaemonApplicationClass

Synopsis: Register alternative `TCustomDaemonApplication` class.

Declaration:
```
procedure RegisterDaemonApplicationClass
                                        (AClass: TCustomDaemonApplicationClass)
```

Visibility: default

Description: `RegisterDaemonApplicationClass` can be used to register an alternative TCustomDaemonApplication (319) descendent which will be used when creating the global Application (315) instance. Only the last registered class pointer will be used.

See also: TCustomDaemonApplication (319), Application (315)

13.4.4 RegisterDaemonClass

Synopsis: Register daemon.

Declaration:
```
procedure RegisterDaemonClass(AClass: TCustomDaemonClass)
```

Visibility: default

Description: `RegisterDaemonClass` must be called for each TCustomDaemon (317) descendent that is used in the class: the class pointer and class name are used by the TCustomDaemonMapperClass (312) class to create a `TCustomDaemon` instance when a daemon is required.

See also: TCustomDaemonMapperClass (312), TCustomDaemon (317)

13.4.5 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 (319) code, so registering an alternative mapping class should happen in the initialization section of the application units.

See also: TCustomDaemonApplication (319), TCustomDaemonMapperClass (312)

13.5 EDaemon

13.5.1 Description

`EDaemon` is the exception class used by all code in the DaemonApp unit.

See also: DaemonError (315)

13.6 TCustomDaemon

13.6.1 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 (327) can be designed to implement the behaviour of the daemon.

See also: TDaemon (327), TDaemonDef (334), TDaemonController (331), TDaemonApplication (331)

13.6.2 Method overview

Page	Method	Description
317	CheckControlMessages	
317	LogMessage	Log a message to the system log.
318	ReportStatus	Report the current status to the operating system.

13.6.3 Property overview

Page	Properties	Access	Description
319	Controller	r	`TDaemonController` instance controlling this daemon instance.
318	DaemonThread	r	Thread in which daemon is running.
318	Definition	r	The definition used to instantiate this daemon instance.
319	Logger	r	`TEventLog` instance used to send messages to the system log.
319	Status	rw	Current status of the daemon.

13.6.4 TCustomDaemon.CheckControlMessages

Synopsis:

Declaration: `procedure CheckControlMessages(Wait: Boolean)`

Visibility: public

Description:

13.6.5 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 (573) instance is used to actually send messages to the system log.

The message is sent with an 'error' flag (using TEventLog.Error (577)).

Errors: None.

See also: ReportStatus (318)

13.6.6 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 (317)

13.6.7 TCustomDaemon.Definition

Synopsis: The definition used to instantiate this daemon instance.

Declaration: `Property Definition : TDaemonDef`

Visibility: public

Access: Read

Description: `Definition` is the TDaemonDef (334) 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 (334)

13.6.8 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 (319)

13.6.9 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 (318)

13.6.10 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 (318)

13.6.11 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 (573) instance used to send messages to the system log. It is used by the LogMessage (317) call, but is accessible through the `Logger` property in case more configurable logging is needed than offered by `LogMessage`.

See also: LogMessage (317), TEventLog (573)

13.7 TCustomDaemonApplication

13.7.1 Description

`TCustomDaemonApplication` is a TCustomApplication (296) 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 (327) instances by checking the TCustomDaemonMapperClass (312) instance that contains the daemon maps.

See also: TCustomApplication (296), TCustomDaemonMapperClass (312)

13.7.2 Method overview

Page	Method	Description
320	Create	Constructor for the class instance.
321	CreateDaemon	Create daemon instance.
322	CreateForm	Create a component.
320	Destroy	Clean up the `TCustomDaemonApplication` instance.
321	InstallDaemons	Install all daemons.
321	RunDaemons	Run all daemons.
320	ShowException	Show an exception.
322	ShowHelp	Display a help message.
321	StopDaemons	Stop all daemons.
322	UnInstallDaemons	Uninstall all daemons.

13.7.3 Property overview

Page	Properties	Access	Description
324	AutoRegisterMessageFile	rw	Automatically register the message file.
323	EventLog	r	Event logger instance.
323	GuiHandle	rw	Handle of GUI loop main application window handle.
323	GUIMainLoop	rw	GUI main loop callback.
322	OnRun	rw	Event executed when the daemon is run.
323	RunMode	r	Application mode.

13.7.4 TCustomDaemonApplication.Create

Synopsis: Constructor for the class instance.

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

Visibility: public

Description: Constructor for the class instance.

13.7.5 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 (323)

13.7.6 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.

13.7.7 TCustomDaemonApplication.CreateDaemon

Synopsis: Create daemon instance.

Declaration: `function CreateDaemon(DaemonDef: TDaemonDef) : TCustomDaemon`

Visibility: public

Description: `CreateDaemon` is called whenever a TCustomDaemon (317) instance must be created from a TDaemonDef (334) daemon definition, passed in `DaemonDef`. It initializes the `TCustomDaemon` instance, and creates a controller instance of type TDaemonController (331) to control the daemon. Finally, it assigns the created daemon to the TDaemonDef.Instance (336) property.

Errors: In case of an error, an exception may be raised.

See also: TDaemonController (331), TCustomDaemon (317), TDaemonDef (334), TDaemonDef.Instance (336)

13.7.8 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 (333), TCustomDaemonApplication.UnInstallDaemons (322), TCustomDaemonApplication.RunDaemons (321)

13.7.9 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 (322), TCustomDaemonApplication.RunDaemons (321), TCustomDaemonApplication.StopDaemons (321)

13.7.10 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 (322), TCustomDaemonApplication.InstallDaemons (321), TCustomDaemonApplication.StopDaemons (321)

13.7.11 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 (321), TCustomDaemonApplication.InstallDaemons (321), TCustomDaemonApplication.StopDaemons (321)

13.7.12 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.

13.7.13 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 (321)

13.7.14 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 (321), TCustomDaemonApplication.InstallDaemons (321), TCustomDaemonApplication.UnInstallDaemons (322)

13.7.15 TCustomDaemonApplication.EventLog

Synopsis: Event logger instance.

Declaration: `Property EventLog : TEventLog`

Visibility: public

Access: Read

Description: `EventLog` is the TEventLog (573) 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 (573)

13.7.16 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 (323)

13.7.17 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 (323) procedure exists, and the application creates and runs a message loop by itself.

See also: GUIMainLoop (323)

13.7.18 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 (321), RunDaemons (321) or UnInstallDaemons (322) is called.

See also: InstallDaemons (321), RunDaemons (321), UnInstallDaemons (322)

13.7.19 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.

13.8 TCustomDaemonMapper

13.8.1 Description

The `TCustomDaemonMapper` class is responsible for mapping a daemon definition to an actual `TDaemon` instance. It maintains a TDaemonDefs (338) 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 (334) definitions and the actual TDaemon (327) instances that will be used to run the daemons.

The `TCustomDaemonMapper` class has no support for streaming. The TDaemonMapper (340) class has support for streaming (and hence visual designing).

See also: TDaemon (327), TDaemonDef (334), TDaemonDefs (338), TDaemonMapper (340)

13.8.2 Method overview

Page	Method	Description
324	Create	Create a new instance of `TCustomDaemonMapper`.
325	Destroy	Clean up and destroy a `TCustomDaemonMapper` instance.

13.8.3 Property overview

Page	Properties	Access	Description
325	DaemonDefs	rw	Collection of daemons.
325	OnCreate	rw	Event called when the daemon mapper is created.
326	OnDestroy	rw	Event called when the daemon mapper is freed.
326	OnInstall	rw	Event called when the daemons are installed.
326	OnRun	rw	Event called when the daemons are executed.
326	OnUnInstall	rw	Event called when the daemons are uninstalled.

13.8.4 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 (338) 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 (338), TCustomDaemonApplication (319), TCustomDaemonMapper.Destroy (325)

13.8.5 TCustomDaemonMapper.Destroy

Synopsis: Clean up and destroy a `TCustomDaemonMapper` instance.

Declaration: `destructor Destroy;  Override`

Visibility: public

Description: `Destroy` frees the DaemonDefs (325) collection and calls the inherited destructor.

See also: TDaemonDefs (338), TCustomDaemonMapper.Create (324)

13.8.6 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 (319)

13.8.7 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 (326), OnUnInstall (326), OnCreate (325), OnDestroy (326)

13.8.8 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 (325) event.

See also: OnCreate (325), OnInstall (326), OnUnInstall (326), OnCreate (325)

13.8.9 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 (326), OnUnInstall (326), OnCreate (325), OnDestroy (326)

13.8.10 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 (326), OnUnInstall (326), OnCreate (325), OnDestroy (326)

13.8.11 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 (326), OnInstall (326), OnCreate (325), OnDestroy (326)

13.9 TDaemon

13.9.1 Description

`TDaemon` is a TCustomDaemon (317) 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 (318)), which is not the main program thread.

See also: TCustomDaemon (317), TDaemonController (331)

13.9.2 Property overview

Page	Properties	Access	Description
330	AfterInstall	rw	Called after the daemon was installed.
330	AfterUnInstall	rw	Called after the daemon is uninstalled.
329	BeforeInstall	rw	Called before the daemon will be installed.
330	BeforeUnInstall	rw	Called before the daemon is uninstalled.
327	Definition		
328	OnContinue	rw	Daemon continue.
330	OnControlCode	rw	Called when a control code is received for the daemon.
331	OnControlCodeEvent	rw	
329	OnExecute	rw	Daemon execute event.
328	OnPause	rw	Daemon pause event.
329	OnShutDown	rw	Daemon shutdown.
327	OnStart	rw	Daemon start event.
328	OnStop	rw	Daemon stop event.
327	Status		

13.9.3 TDaemon.Definition

Declaration: `Property Definition :`

Visibility: public

Access:

13.9.4 TDaemon.Status

Declaration: `Property Status :`

Visibility: public

Access:

13.9.5 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 (318) 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 (329) event handler, so the event handler can return at once.

See also: TDaemon.OnStop (328), TDaemon.OnExecute (329), TDaemon.OnContinue (328), ReportStatus (318)

13.9.6 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 (318) method.

If a thread was started in the OnExecute (329) event, this is the place where the thread should be stopped.

See also: TDaemon.OnStart (327), TDaemon.OnPause (328), ReportStatus (318)

13.9.7 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 (318) method.

If a thread was started in the OnExecute (329) event, this is the place where the thread's execution should be suspended.

See also: TDaemon.OnStop (328), TDaemon.OnContinue (328), ReportStatus (318)

13.9.8 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 (318) method.

If a thread was started in the OnExecute (329) event and it was suspended in a OnPause (327) event, this is the place where the thread's executed should be resumed.

See also: TDaemon.OnStart (327), TDaemon.OnPause (328), ReportStatus (318)

13.9.9 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 (329), this is the place where the thread's executed should be stopped or the thread freed from memory.

See also: TDaemon.OnStart (327), TDaemon.OnPause (328), ReportStatus (318)

13.9.10 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 (327), TDaemon.OnStop (328)

13.9.11 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 (330), BeforeUnInstall (330), AfterUnInstall (330)

13.9.12 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 (329), BeforeUnInstall (330), AfterUnInstall (330)

13.9.13 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 (329), AfterInstall (330), AfterUnInstall (330)

13.9.14 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 (329), AfterInstall (330), BeforeUnInstall (330)

13.9.15 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 (309)

13.9.16 TDaemon.OnControlCodeEvent

Synopsis:

Declaration: `Property OnControlCodeEvent : TCustomControlCodeEvEvent`

Visibility: published

Access: Read,Write

Description:

13.10 TDaemonApplication

13.10.1 Description

`TDaemonApplication` is the default TCustomDaemonApplication (319) descendent that is used to run the daemon application. It is possible to register an alternative `TCustomDaemonApplication` class (using RegisterDaemonApplicationClass (316)) to run the application in a different manner.

See also: TCustomDaemonApplication (319), RegisterDaemonApplicationClass (316)

13.11 TDaemonController

13.11.1 Description

`TDaemonController` is a class that is used by the TDaemonApplication (331) 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 (317), TDaemonApplication (331)

13.11.2 Method overview

Page	Method	Description
333	Controller	Controller.
332	Create	Create a new instance of the `TDaemonController` class.
332	Destroy	Free a `TDaemonController` instance.
332	Main	Daemon main entry point.
333	ReportStatus	Report the status to the operating system.
332	StartService	Start the service.

13.11.3 Property overview

Page	Properties	Access	Description
334	CheckPoint	r	Send checkpoint signal to the operating system.
333	Daemon	r	Daemon instance this controller controls.
334	LastStatus	r	Last reported status.
333	Params	r	Parameters passed to the daemon.

13.11.4 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 (331) instance, and `AOwner` will be set to the global TDaemonApplication (331) instance.

See also: TDaemonApplication (331), Destroy (332)

13.11.5 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 (332)

13.11.6 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 (332)

13.11.7 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 (341)

13.11.8 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 (330)

13.11.9 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 (318), TDaemonController.LastStatus (334)

13.11.10 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.

13.11.11 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 (332) method.

13.11.12 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 (333)

13.11.13 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 (333)

13.12 TDaemonDef

13.12.1 Description

`TDaemonDef` contains the definition of a daemon in the application: The name of the daemon, which TCustomDaemon (317) 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 (331), TDaemon (327)

13.12.2 Method overview

Page	Method	Description
335	Create	Create a new `TDaemonDef` instance.
335	Destroy	Free a `TDaemonDef` from memory.

13.12.3 Property overview

Page	Properties	Access	Description
335	DaemonClass	r	`TDaemon` class to use for this daemon.
336	DaemonClassName	rw	Name of the `TDaemon` class to use for this daemon.
336	Description	rw	Description of the daemon.
337	DisplayName	rw	Displayed name of the daemon (service).
337	Enabled	rw	Is the daemon enabled or not.
336	Instance	rw	Instance of the daemon class.
338	LogStatusReport	rw	Log the status report to the system log.
336	Name	rw	Name of the daemon (service).
338	OnCreateInstance	rw	Event called when a daemon in instantiated.
337	Options	rw	Service options.
337	RunArguments	rw	Additional command-line arguments when running daemon.
338	WinBindings	rw	Windows-specific bindings (windows only).

13.12.4 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 (338)

13.12.5 TDaemonDef.Destroy

Synopsis: Free a `TDaemonDef` from memory.

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

Visibility: public

Description: `Destroy` removes the `TDaemonDef` from memory.

13.12.6 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 (336).

See also: DaemonClassName (336), TDaemonMapper (340)

13.12.7 TDaemonDef.Instance

Synopsis: Instance of the daemon class.

Declaration: `Property Instance : TCustomDaemon`

Visibility: public

Access: Read,Write

Description: `Instance` points to the TDaemon (327) instance that is used when the service is in operation at runtime.

See also: TDaemonDef.DaemonClass (335)

13.12.8 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 (319) creates an instance of the daemon.

See also: TDaemonDef.Instance (336), TDaemonDef.DaemonClass (335), RegisterDaemonClass (316), TCustomDaemon (317), TCustomDaemonApplication (319)

13.12.9 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 (337)

13.12.10 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.

13.12.11 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 (336)

13.12.12 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.

13.12.13 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.

13.12.14 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.

13.12.15 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 (346)

13.12.16 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.

13.12.17 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 (318)

13.13 TDaemonDefs

13.13.1 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 (324) instance will create a collection and maintain it.

See also: TCustomDaemonMapper (324), TDaemonDef (334)

13.13.2 Method overview

Page	Method	Description
339	Create	Create a new instance of a `TDaemonDefs` collection.
340	DaemonDefByName	Find and return instance of daemon definition with given name.
339	FindDaemonDef	Find and return instance of daemon definition with given name.
339	IndexOfDaemonDef	Return index of daemon definition.

13.13.3 Property overview

Page	Properties	Access	Description
340	Daemons	rw	Indexed access to `TDaemonDef` instances.

13.13.4 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 (324) instance will create a collection and maintain it.

See also: TDaemonDef (334)

13.13.5 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 (334) 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 (339), TDaemonDefs.DaemonDefByName (340)

13.13.6 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 (334) instance whose name matches `DaemonName`. If no definition is found, `Nil` is returned.

See also: TDaemonDefs.IndexOfDaemonDef (339), TDaemonDefs.DaemonDefByName (340)

13.13.7 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 (334) instance whose name matches `DaemonName`. If no definition is found, an EDaemon (316) exception is raised.

The FindDaemonDef (339) call does not raise an error, but returns `Nil` instead.

Errors: If no definition is found, an EDaemon (316) exception is raised.

See also: TDaemonDefs.IndexOfDaemonDef (339), TDaemonDefs.FindDaemonDef (339)

13.13.8 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 (334)

13.14 TDaemonMapper

13.14.1 Description

`TDaemonMapper` is a direct descendent of TCustomDaemonMapper (324), 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 (341) constructor can be used instead.

See also: CreateNew (341), Create (341)

13.14.2 Method overview

Page	Method	Description
341	Create	Create a new `TDaemonMapper` instance and initializes it from streamed resources.
341	CreateNew	Create a new `TDaemonMapper` instance without initialization.

13.14.3 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 (341) 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 (341)

13.14.4 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 (341)

13.15 TDaemonThread

13.15.1 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 (331) class will take care of this.

See also: TDaemonController (331), TDaemon (327)

13.15.2 Method overview

Page	Method	Description
342	CheckControlMessage	Check if a control message has arrived.
343	ContinueDaemon	Continue the daemon.
342	Create	Create a new thread.
342	Execute	Run the daemon.
343	InterrogateDaemon	Report the daemon status.
343	PauseDaemon	Pause the daemon.
343	ShutDownDaemon	Shut down daemon.
342	StopDaemon	Stops the daemon.

13.15.3 Property overview

Page	Properties	Access	Description
344	Daemon	r	Daemon instance.

13.15.4 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 (344)

13.15.5 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 (327)

13.15.6 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.

13.15.7 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 (344), TDaemonThread.PauseDaemon (343), TDaemonThread.ShutDownDaemon (343), TCustomDaemon (317), TThread.Terminate (**??**)

13.15.8 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 (317) instance, and calling `Suspend` to suspend the thread. It returns `True` if the attempt was successful.

See also: TDaemonThread.StopDaemon (342), TDaemonThread.ContinueDaemon (343), TDaemonThread.ShutDownDaemon (343), TCustomDaemon (317), TThread.Suspend (**??**)

13.15.9 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 (317) instance. It returns `True` if the attempt was successful.

See also: TDaemonThread.Daemon (344), TDaemonThread.StopDaemon (342), TDaemonThread.PauseDaemon (343), TDaemonThread.ShutDownDaemon (343), TCustomDaemon (317)

13.15.10 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 (317) instance. The thread is terminated in this method.

See also: TDaemonThread.StopDaemon (342), TDaemonThread.PauseDaemon (343), TDaemonThread.ContinueDaemon (343), TCustomDaemon (317), TThread.Terminate (**??**)

13.15.11 TDaemonThread.InterrogateDaemon

Synopsis: Report the daemon status.

Declaration: `function InterrogateDaemon : Boolean; Virtual`

Visibility: public

Description: `InterrogateDaemon` simply calls TCustomDaemon.ReportStatus (318) for the daemon that is running in this thread. It always returns `True`.

See also: TCustomDaemon.ReportStatus (318)

13.15.12 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 (327)

13.16 TDependencies

13.16.1 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 (345) instances.

See also: TDependency (345)

13.16.2 Method overview

Page	Method	Description
344	Create	Create a new instance of a `TDependencies` collection.

13.16.3 Property overview

Page	Properties	Access	Description
344	Items	rw	Default property override.

13.16.4 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.

13.16.5 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 (345).

See also: TDependency (345)

13.17 TDependency

13.17.1 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 (344), TDaemonDef (334)

13.17.2 Method overview

Page	Method	Description
345	Assign	Assign `TDependency` instance to another.

13.17.3 Property overview

Page	Properties	Access	Description
345	IsGroup	rw	Name refers to a service group.
345	Name	rw	Name of the service.

13.17.4 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.

13.17.5 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 (345)

13.17.6 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 (345)

13.18 TWinBindings

13.18.1 Description

`TWinBindings` contains windows-specific properties for the daemon definition (in TDaemonDef.WinBindings (338)). If the daemon should not run on Windows, then the properties can be ignored.

See also: TDaemonDef (334), TDaemonDef.WinBindings (338)

13.18.2 Method overview

Page	Method	Description
347	Assign	Copies all properties.
346	Create	Create a new `TWinBindings` instance.
346	Destroy	Remove a `TWinBindings` instance from memory.

13.18.3 Property overview

Page	Properties	Access	Description
350	AcceptedCodes	rw	
347	Dependencies	rw	Service dependencies.
347	ErrCode	rw	Service specific error code.
349	ErrorSeverity	rw	Error severity in case of startup failure.
348	GroupName	rw	Service group name.
349	IDTag	rw	Location in the service group.
348	Password	rw	Password for service startup.
349	ServiceType	rw	Type of service.
348	StartType	rw	Service startup type.
348	UserName	rw	Username to run service as.
349	WaitHint	rw	Timeout wait hint.
347	Win32ErrCode	rw	General windows error code.

13.18.4 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 (334), TDaemonDef.WinBindings (338), TWinBindings.Dependencies (347)

13.18.5 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 (347), TWinBindings.Create (346)

13.18.6 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.

13.18.7 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 (318) to the windows service manager. If it is zero, then the contents of Win32ErrCode (347) are reported. If it is nonzero, then the windows-errorcode is set to `ERROR_SERVICE_SPECIFIC_ERROR`.

See also: TWinBindings.Win32ErrCode (347)

13.18.8 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 (318) to the windows service manager. It is sent if ErrCode (347) is zero.

See also: ErrCode (347)

13.18.9 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.

13.18.10 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 (345)

13.18.11 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 (348)

13.18.12 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 (348) property.

This property is only used during installation of the service.

See also: Password (348)

13.18.13 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.

13.18.14 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 (318)

13.18.15 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.

13.18.16 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.

13.18.17 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.

13.18.18 TWinBindings.AcceptedCodes

Synopsis:

Declaration: `Property AcceptedCodes : TWinControlCodes`

Visibility: published

Access: Read,Write

Description:

Chapter 14

Reference for unit 'DB'

14.1 Used units

Table 14.1: Used units by unit 'DB'

Name	Page
Classes	??
FmtBCD	??
MaskUtils	730
System	??
sysutils	??
Variants	??

14.2 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 (409) representing a set of records from a database, TField (462) which represents the contents of a field in a record, TDatasource (449) which acts as an event distributor on behalf of a dataset and TParams (544) which can be used to parameterize queries. The databases connections themselves are abstracted in the TDatabase (400) class.

14.3 Constants, types and variables

14.3.1 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 (462) descendent class to use when a TDataset instance needs to create fields based on the TFieldDefs (494) 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 (437) 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 (437) 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 (354) 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.

14.3.2 Types

```
LargeInt = Int64
```

Large (64-bit) integer.

```
PBookmarkFlag = ^TBookmarkFlag
```

PBookmarkFlag is a convenience type, defined for internal use in TDataset (409) or one of it's descendents.

```
PBufferList = ^TBufferList
```

PBufferList is a pointer to a structure of type TBufferList (354). 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 14.2: 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 (409) method. It is of type `TBytes`, the contents of the buffer is determined by the actual `TDataset` descendent.

```
TBookmarkFlag = (bfCurrent,bfBOF,bfEOF,bfInserted)
```

Table 14.3: 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 (409) 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 (431) 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 (354) 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 (409) 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 14.4: 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 (356) 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 (400) class.

```
TDataChangeEvent = procedure(Sender: TObject; Field: TField) of
  object
```

`TDataChangeEvent` is the event handler prototype for the TDatasource.OnDataChange (452) 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 14.5: 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 (449) instances connected to a TDataset (409) 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 (409) 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 (446), TDataset.OnPostError (447) and TDataset.OnDeleteError (446) 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 (355).

```
TDataSetNotifyEvent = procedure(DataSet: TDataSet) of object
```

`TDatasetNotifyEVent` is used in most of the TDataset (409) 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 14.6: 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 (409) 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 (455)

```
TFieldAttribute = (faHiddenCol,faReadonly,faRequired,faLink,faUnNamed
  ,
                   faFixed)
```

Table 14.7: 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 (492) property of TFieldDef (488).

```
TFieldAttributes = Set of TFieldAttribute
```

TFieldAttributes is used in the TFieldDef.Attributes (492) property to denote additional attributes of the underlying field.

```
TFieldChars = Set of Char
```

TFieldChars is a type used in the TField.ValidChars (479) 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 (434) items class.

```
TFieldDefsClass = Class of TFieldDefs
```

TFieldDefClass is used to be able to customize the actual TDataset.FieldDefs (434) class used in a TDataset (409) descendent.

```
TFieldGetTextEvent = procedure(Sender: TField; var aText: string;
  DisplayText: Boolean) of object
```

TFieldGetTextEvent is the prototype for the TField.OnGetText (487) 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 14.8: 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 (359) 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 (462) 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 (437);

```
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 14.9: 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 (462) underlying data, in the DataType (475) property.

```
TFilterOption = (foCaseInsensitive,foNoPartialCompare)
```

Table 14.10: 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 (362) set is used in the TDataset.FilterOptions (439) 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 (439) property. The available values are described in the TFilterOption (362) type.

```
TFilterRecordEvent = procedure(DataSet: TDataSet; var Accept: Boolean
  )
                                    of object
```

`TFilterRecordEvent` is the prototype for the TDataset.OnFilterRecord (447) 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 14.11: 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 (409) 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 14.12: 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 (409) when they have to communicate the result of the GetRecord operation back to the TDataset record.

```
TIndexOption = (ixPrimary,ixUnique,ixDescending,ixCaseInsensitive
  ,
                ixExpression,ixNonMaintained)
```

Table 14.13: 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 (363) set type to describe all properties of an index definition as in TIndexDef (509).

```
TIndexOptions = Set of TIndexOption
```

TIndexOptions contains the set of properties that an index can have. It is used in the TIndexDef.Options (511) property to describe all properties of an index definition as in TIndexDef (509).

```
TLocateOption = (loCaseInsensitive,loPartialKey)
```

Table 14.14: 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 (426) 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 (426) 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 (399) 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 (544) collection.

```
TParamStyle = (psInterbase,psPostgreSQL,psSimulated)
```

Table 14.15: 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 (547) 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 14.16: 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 (530) 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 (364) values.

```
TProviderFlag = (pfInUpdate,pfInWhere,pfInKey,pfHidden,
  pfRefreshOnInsert,pfRefreshOnUpdate)
```

Table 14.17: 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 (409) has one or more of these flags.

```
TProviderFlags = Set of TProviderFlag
```

`TProviderFlags` is used for the TField.ProviderFlags (486) 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 14.18: 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 (376) 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 (409) 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 (366)

```
TResolverResponse = (rrSkip,rrAbort,rrMerge,rrApply,rrIgnore)
```

Table 14.19: 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 14.20: 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 (409) navigation and data manipulation methods such as the TDataset.Refresh (429) method when they need to reset the cursor position in the dataset's buffer.

```
TSQLParseOption = (spoCreate,spoEscapeSlash,spoEscapeRepeat,spoUseMacro
  )
```

Table 14.21: 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 (366) 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 14.22: 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 (409) 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 14.23: 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 14.24: 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 14.25: 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 (368) values.

14.3.3 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.

14.4 Procedures and functions

14.4.1 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 (**??**)

14.4.2 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 (371) exception, passing it `Msg`. If `Comp` is specified, the name of the component is prepended to the message.

See also: DatabaseErrorFmt (369), EDatabaseError (371)

14.4.3 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 (371) 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 (369), EDatabaseError (371)

14.4.4 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 (360), TDateTimeRec (357), DateTimeToDateTimeRec (370)

14.4.5 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 (360), TDateTimeRec (357), DateTimeRecToDateTime (370)

14.4.6 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 (**??**)

14.4.7 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 (448) as an enumerator for a TDataset (409)

See also: TDataSetEnumerator (448), TDataset (409)

14.4.8 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 (497)

14.4.9 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 (547)

14.5 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.

14.6 EDatabaseError

14.6.1 Description

`EDatabaseError` is the base class from which database-related exception classes should derive. It is raised by the DatabaseError (369) call.

See also: DatabaseError (369), DatabaseErrorFmt (369)

14.7 EUpdateError

14.7.1 Description

`EupdateError` is an exception used by the TProvider database support. It should never be raised directly.

See also: EDatabaseError (371)

14.7.2 Method overview

Page	Method	Description
372	Create	Create a new EUpdateError instance.
372	Destroy	Free the EupdateError instance.

14.7.3 Property overview

Page	Properties	Access	Description
373	Context	r	Context in which exception occurred.
373	ErrorCode	r	Numerical error code.
373	OriginalException	r	Originally raised exception.
373	PreviousError	r	Previous error number.

14.7.4 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 (371)

14.7.5 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 (373)

14.7.6 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 (373), EUpdateError.ErrorCode (373), EUpdateError.PreviousError (373)

14.7.7 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 (373), EUpdateError.Context (373), EUpdateError.PreviousError (373)

14.7.8 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 (371)

14.7.9 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 (373), EUpdateError.Context (373), EUpdateError.OriginalException (373)

14.8 IProviderSupport

14.8.1 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 (409) 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 (409)

14.8.2 Method overview

Page	Method	Description
374	PSEndTransaction	End an active transaction.
374	PSExecute	Execute the current command-text.
375	PSExecuteStatement	Execute a SQL statement.
375	PSGetAttributes	Get a list of attributes (metadata).
375	PSGetCommandText	Return the SQL command executed for getting data.
376	PSGetCommandType	Return SQL command type.
376	PSGetDefaultOrder	Default order index definition.
376	PSGetIndexDefs	Return a list of index definitions.
376	PSGetKeyFields	Return a list of key fields in the dataset.
377	PSGetParams	Get the parameters in the commandtext.
377	PSGetQuoteChar	Quote character for quoted strings.
377	PSGetTableName	Name of database table which must be updated.
377	PSGetUpdateException	Transform exception to `UpdateError`.
378	PSInTransaction	Is the dataset in an active transaction.
378	PSIsSQLBased	Is the dataset SQL based.
378	PSIsSQLSupported	Can the dataset support SQL statements.
378	PSReset	Position the dataset on the first record.
379	PSSetCommandText	Set the command-text of the dataset.
379	PSSetParams	Set the parameters for the command text.
379	PSStartTransaction	Start a new transaction.
379	PSUpdateRecord	Update a record.

14.8.3 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 (351) returns `True`). If `ACommit` is `True` then the transaction is committed, else it is rolled back.

See also: PSInTransaction (351), PSStartTransaction (351)

14.8.4 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 (351).

See also: PSGetCommandText (351), PSExecuteStatement (351)

14.8.5 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 (351) returns the same value before and after a call to `PSExecuteStatement`.

See also: PSGetCommandText (351), PSSetCommandText (351), PSExecuteStatement (351)

14.8.6 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 (351)

14.8.7 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 (351) function is called (for a TSQLQuery this would be the SQL property) or when the dataset is opened.

See also: PSExecute (351), PSSetCommandText (351)

14.8.8 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 (351)). The list of possible command types is enumerated in TPSCommandType (365).

See also: PSGetCommandText (351), TPSCommandType (365), PSExecute (351)

14.8.9 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 (351)) that represents the default sort order.

See also: PSGetIndexDefs (351), PSGetKeyFields (351)

14.8.10 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 (351), PSGetKeyFields (351)

14.8.11 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 (462) property.

See also: PSGetIndexDefs (351), PSGetDefaultOrder (351), TField.ProviderOptions (462), TProviderFlags (365)

14.8.12 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 (351)). This is usually the `Params` property of a TDataset (409) descendant.

See also: PSGetCommandText (351), PSSetParams (351)

14.8.13 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 (351)

14.8.14 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 (351)

14.8.15 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 (372) exception. The previous `EUpdateError` exception in the update batch is passed in `Prev`.

See also: EUpdateError (372)

14.8.16 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 (351), PSStartTransaction (351)

14.8.17 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 (351) which indicates whether SQL statements can be executed using PSExecuteCommand (351)

See also: PSIsSQLSupported (351), PSExecuteCommand (351)

14.8.18 IProviderSupport.PSIsSQLSupported

Synopsis: Can the dataset support SQL statements.

Declaration: `function PSIsSQLSupported : Boolean`

Visibility: default

Description: `PSIsSQLSupported` returns `True` if PSExecuteCommand (351) can be used to execute SQL statements on the underlying database.

See also: PSExecuteCommand (351)

14.8.19 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 (423), TDataset.Open (428)

14.8.20 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 (351), PSGetCommandText (351), PSSetParams (351)

14.8.21 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 (351), PSGetParams (351)

14.8.22 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 (351) returned `False`).

See also: PSEndTransaction (351), PSIntransaction (351)

14.8.23 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 (351), PSExecuteCommand (351)

14.9 TArrayField

14.9.1 Method overview

Page	Method	Description
380	Create	

14.9.2 TArrayField.Create

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

Visibility: public

14.10 TAutoIncField

14.10.1 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 (462)

14.10.2 Method overview

Page	Method	Description
380	Create	Create a new instance of the `TAutoIncField` class.

14.10.3 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 (462) class' fields.

See also: TField (462)

14.11 TBCDField

14.11.1 Description

`TBCDField` is the class used when a dataset must manage data of Binary Coded Decimal type. (TField.DataType (475) equals `ftBCD`). It initializes some of the properties of the TField (462) 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 (505) 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 (409), TField (462), TFMTBCDField (505)

14.11.2 Method overview

Page	Method	Description
381	CheckRange	Check whether a values falls within the allowed range.
381	Create	Create a new instance of a TBCDField class.

14.11.3 Property overview

Page	Properties	Access	Description
382	Currency	rw	Does the field represent a currency amount.
382	MaxValue	rw	Maximum value for the field.
383	MinValue	rw	Minimum value for the field.
382	Precision	rw	Precision of the BCD field.
383	Size		Number of decimals after the decimal separator.
382	Value	rw	Value of the field contents as a Currency type.

14.11.4 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 (462) properties to configure the instance for working with BCD data values.

See also: TField (462)

14.11.5 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 (383) and MaxValue (382) properties. If the value lies outside of the allowed range, then False is returned.

See also: MaxValue (382), MinValue (383)

14.11.6 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 (479) property to a currency type field. It returns the same value as the TField.AsCurrency (469) field.

See also: TField.Value (479), TField.AsCurrency (469)

14.11.7 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 (383), 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 (383), TBCDField.Value (382)

14.11.8 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 (476) and TField.Text (478) 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 (476), TField.Text (478)

14.11.9 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 (383) should also be set, because it will also be checked.

See also: TBCDField.MinValue (383), TBCDField.CheckRange (381)

14.11.10 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 (382) should also be set, because it will also be checked.

See also: TBCDField.MaxValue (382), TBCDField.CheckRange (381)

14.11.11 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 (382) field.

See also: TBCDField.Precision (382)

14.12 TBinaryField

14.12.1 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 (462) 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 (391) or TVarBytesField (557) should be created.

See also: TDataset (409), TField (462), TBytesField (391), TVarBytesField (557)

14.12.2 Method overview

Page	Method	Description
384	Create	Create a new instance of a `TBinaryField` class.

14.12.3 Property overview

Page	Properties	Access	Description
384	Size		Size of the binary data.

14.12.4 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 (462)

14.12.5 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 (478)

14.13 TBlobField

14.13.1 Description

`TBLobField` is the class used when a dataset must manage BLOB data. (TField.DataType (475) equals `ftBLOB`). It initializes some of the properties of the TField (462) 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 (525), TWideMemoField (558) or TGraphicField (508)

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 (409), TField (462), TMemoField (525), TWideMemoField (558), TGraphicField (508)

14.13.2 Method overview

Page	Method	Description
385	Clear	Clear the BLOB field's contents.
385	Create	Create a new instance of a `TBlobField` class.
385	IsBlob	Is the field a blob field.
386	LoadFromFile	Load the contents of the field from a file.
386	LoadFromStream	Load the field's contents from stream.
386	SaveToFile	Save field contents to a file.
387	SaveToStream	Save the field's contents to stream.
387	SetFieldType	Set field type.

14.13.3 Property overview

Page	Properties	Access	Description
387	BlobSize	r	Size of the current blob.
388	BlobType	rw	Type of blob.
387	Modified	rw	Has the field's contents been modified.
388	Size		Size of the blob field.
388	Transliterate	rw	Should the contents of the field be transliterated.
388	Value	rw	Return the field's contents as a string.

14.13.4 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 (462) properties to configure the instance for working with BLOB data.

See also: TField (462)

14.13.5 TBlobField.Clear

Synopsis: Clear the BLOB field's contents.

Declaration:
```
procedure Clear;  Override
```

Visibility: public

Description: `Clear` overrides the `TField` implementation of TField.Clear (466). It creates and immediately releases an empty blob stream in write mode, effectively clearing the contents of the BLOB field.

See also: TField.Clear (466), TField.IsNull (477)

14.13.6 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 (467)

14.13.7 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 (386) 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 (386), SaveToFile (386)

14.13.8 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 (387), LoadFromFile (386)

14.13.9 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 (387) 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 (386), SaveToStream (387)

14.13.10 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 (386), LoadFromStream (386)

14.13.11 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 (475)

14.13.12 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 (478), TField.DataSize (475)

14.13.13 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 (386)

14.13.14 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 (472) property.

See also: TField.Value (479), TField.AsString (472)

14.13.15 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 (430) 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 (555), TDataset.Translate (430)

14.13.16 TBlobField.BlobType

Synopsis: Type of blob.

Declaration: `Property BlobType : TBlobType`

Visibility: published

Access: Read,Write

Description: `BlobType` is an alias for TField.DataType (475), but with a restricted set of values. Setting `BlobType` is equivalent to setting the TField.DataType (475) property.

See also: TField.DataType (475)

14.13.17 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 (387) property instead.

See also: BlobSize (387)

14.14 TBooleanField

14.14.1 Description

`TBooleanField` is the field class used by TDataset (409) whenever it needs to manage boolean data (TField.DataType (475) equals `ftBoolean`). It overrides some properties and methods of TField (462) 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 (409), TField (462)

14.14.2 Method overview

Page	Method	Description
389	Create	Create a new instance of the `TBooleanField` class.

14.14.3 Property overview

Page	Properties	Access	Description
390	DisplayValues	rw	Textual representation of the true and false values.
389	Value	rw	Value of the field as a boolean value.

14.14.4 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 (462) properties to configure it for working with boolean values.

See also: TField (462)

14.14.5 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 (479) by `TBooleanField` as a boolean value. It returns the same value as the TField.AsBoolean (469) property.

See also: TField.AsBoolean (469), TField.Value (479)

14.14.6 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 (472), TField.Text (478)

14.15 TByteField

14.15.1 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 (462) 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 (518) and MaxValue (518).

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 (518), MaxValue (518)

14.15.2 Method overview

Page	Method	Description
390	Create	Create new instance of `TByteField`.

14.15.3 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 (518)MaxValue (518) and TField.DataType (475) properties.

See also: MinValue (518), MaxValue (518), TField.DataType (475)

14.16 TBytesField

14.16.1 Description

`TBytesField` is the class used when a dataset must manage data of fixed-size binary type. (TField.DataType (475) equals `ftBytes`). It initializes some of the properties of the TField (462) 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 (409), TField (462), TVarBytesField (557)

14.16.2 Method overview

Page	Method	Description
391	Create	Create a new instance of a `TBytesField` class.

14.16.3 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 (462) properties to configure the instance for working with binary data values.

See also: TField (462)

14.17 TCheckConstraint

14.17.1 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 (393) property , or a constraint enforced by the user on a particular dataset instance stored in TCheck-Constraint.CustomConstraint (392)

See also: TCheckConstraints (393), TCheckConstraint.ImportedConstraint (393), TCheckConstraint.CustomConstraint (392)

14.17.2 Method overview

Page	Method	Description
392	Assign	Assign one constraint to another.

14.17.3 Property overview

Page	Properties	Access	Description
392	CustomConstraint	rw	User-defined constraint.
392	ErrorMessage	rw	Message to display when the constraint is violated.
393	FromDictionary	rw	True if the constraint is imported from a datadictionary.
393	ImportedConstraint	rw	Constraint imported from the database engine.

14.17.4 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 (393), TCheckConstraint.CustomConstraint (392)

14.17.5 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 (409) 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 (392)

See also: TCheckConstraint.ErrorMessage (392)

14.17.6 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 (392)

14.17.7 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 (409) descendents to indicate the source of the constraint, but is otherwise ignored.

See also: TCheckConstraint.ImportedConstraint (393)

14.17.8 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 (409) descendent.

See also: TCheckConstraint.CustomConstraint (392)

14.18 TCheckConstraints

14.18.1 Description

`TCheckConstraints` is a `TCollection` descendent which keeps a collection of TCheckConstraint (391) items. It overrides the Add (394) method to return a `TCheckConstraint` instance.

See also: TCheckConstraint (391)

14.18.2 Method overview

Page	Method	Description
394	Add	Add new `TCheckConstraint` item to the collection.
394	Create	Create a new instance of the `TCheckConstraints` class.

14.18.3 Property overview

Page	Properties	Access	Description
394	Items	rw	Indexed access to the items in the collection.

14.18.4 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 (409) 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 (391) class pointer.

See also: TCheckConstraint (391), TDataset (409)

14.18.5 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 (391) instance to the collection. it returns the newly added instance.

See also: TCheckConstraint (391), #rtl.classes.TCollection.Add (**??**)

14.18.6 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 (**??**)

14.19 TCurrencyField

14.19.1 Description

`TCurrencyField` is the field class used by TDataset (409) when it needs to manage currency-valued data.(TField.Datatype (475) equals `ftCurrency`). It simply sets some Tfield (462) 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 (462), TDataset (409)

14.19.2 Method overview

Page	Method	Description
395	Create	Create a new instance of a `TCurrencyField`.

14.19.3 Property overview

Page	Properties	Access	Description
395	Currency		Is the field a currency field.

14.19.4 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 (395)) to be able to work with currency data.

See also: TField (462), TCurrencyField.Currency (395)

14.19.5 TCurrencyField.Currency

Synopsis: Is the field a currency field.

Declaration: `Property Currency :`

Visibility: published

Access:

Description: `Currency` is inherited from TFloatField.Currency (504) 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 (504)

14.20 TCustomConnection

14.20.1 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 (396)) and to deactivate the connection (TCustomConnection.Close (396)), plus a property to inspect the state (Connected (397)) of the connected.

See also: TCustomConnection.Open (396), TCustomConnection.Close (396), TCustomConnection.Connected (397)

14.20.2 Method overview

Page	Method	Description
396	Close	Close the connection.
396	Destroy	Remove the `TCustomconnection` instance from memory.
396	Open	Makes the connection to the server.

14.20.3 Property overview

Page	Properties	Access	Description
398	AfterConnect	rw	Event triggered after a connection is made.
398	AfterDisconnect	rw	Event triggered after a connection is closed.
399	BeforeConnect	rw	Event triggered before a connection is made.
399	BeforeDisconnect	rw	Event triggered before a connection is closed.
397	Connected	rw	Is the connection established or not.
397	DataSetCount	r	Number of datasets connected to this connection.
397	DataSets	r	Datasets linked to this connection.
398	LoginPrompt	rw	Should the `OnLogin` be triggered.
399	OnCloseError	rw	
399	OnLogin	rw	Event triggered when a login prompt is shown.

14.20.4 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 (399) 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 (398) event is triggered.

Calling the `Close` method is equivalent to setting the Connected (397) 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 (371) exception will be raised.

See also: TCustomConnection.BeforeDisconnect (399), TCustomConnection.AfterDisconnect (398), TCustomConnection.Open (396), TCustomConnection.Connected (397)

14.20.5 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 (396)

14.20.6 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 (399) event. If an exception is raised during the execution of that event handler, the connect process is aborted. If LoginPrompt (398) is `True`, the OnLogin (399) event handler is called. Only after this event, the connection is actually established. After the connection was established, the AfterConnect (398) event is triggered.

Calling the `Open` method is equivalent to setting the Connected (397) 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 (399), TCustomConnection.LoginPrompt (398), TCustomConnection.OnLogin (399), TCustomConnection.AfterConnect (398), TCustomConnection.Connected (397)

14.20.7 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 (397) array property. As implemented in `TCustomConnection`, this property is always zero. Descendent classes implement the actual count.

See also: TDataset (409), TCustomConnection.Datasets (397)

14.20.8 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 (397).

See also: DatasetCount (397)

14.20.9 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 (396), or to `False` to break it (equivalent to calling TCustomConnection.Close (396).

See also: TCustomConnection.Open (396), TCustomConnection.Close (396)

14.20.10 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 (399)

14.20.11 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 (396). It can be used to open datasets, or indicate a connection status change.

See also: TCustomConnection.Open (396), TCustomConnection.BeforeConnect (399), TCustomConnection.OnLogin (399)

14.20.12 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 (396). It can be used for instance to indicate a connection status change.

See also: TCustomConnection.Close (396), TCustomConnection.BeforeDisconnect (399)

14.20.13 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 (396). 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 (396), TCustomConnection.AfterConnect (398), TCustomConnection.OnLogin (399)

14.20.14 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 (396). 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 (396), TCustomConnection.AfterDisconnect (398)

14.20.15 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 (398) property is `True`, after the TCustomConnection.BeforeConnect (399) event, but before the connection is actually established.

See also: TCustomConnection.BeforeConnect (399), TCustomConnection.LoginPrompt (398), TCustomConnection.Open (396)

14.20.16 TCustomConnection.OnCloseError

Declaration: `Property OnCloseError : TCloseErrorEvent`

Visibility: published

Access: Read,Write

14.21 TDatabase

14.21.1 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 (409), 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 (403) 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 (409), TDatabase (400)

14.21.2 Method overview

Page	Method	Description
401	CloseDataSets	Close all connected datasets.
401	CloseTransactions	End all transactions.
400	Create	Initialize a new `TDatabase` class instance.
401	Destroy	Remove a `TDatabase` instance from memory.
402	EndTransaction	End an active transaction.
401	StartTransaction	Start a new transaction.

14.21.3 Property overview

Page	Properties	Access	Description
403	Connected	rw	Is the database connected.
403	DatabaseName	rw	Database name or path.
402	Directory	rw	Directory for the database.
403	IsSQLBased	r	Is the database SQL based.
403	KeepConnection	rw	Should the connection be kept active.
404	Params	rw	Connection parameters.
402	TransactionCount	r	Number of transaction components connected to this database.
402	Transactions	r	Indexed access to all transaction components connected to this database.

14.21.4 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 (455), TDBTransaction (456), TDatabase.Destroy (401)

14.21.5 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 (401)

14.21.6 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 (396), TDatabase.CloseTransactions (401)

14.21.7 TDatabase.CloseTransactions

Synopsis: End all transactions.

Declaration: `procedure CloseTransactions`

Visibility: public

Description: `CloseTransaction` calls TDBTransaction.EndTransaction (456) on all connected transactions. It is called automatically when the connection is closed, after all datasets are closed.

See also: TCustomConnection.Close (396), TDatabase.CloseDatasets (401)

14.21.8 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 (456) component instead and invoke the TDBTransaction.StartTransaction (456) method.

See also: TDBTransaction (456), TDBTransaction.StartTransaction (456)

14.21.9 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 (456) component instead and invoke the TDBTransaction.EndTransaction (456) method.

See also: TDBTransaction (456), TDBTransaction.EndTransaction (456)

14.21.10 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 (402) array property.

See also: TDatabase.Transactions (402)

14.21.11 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 (402)

14.21.12 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 (400).

See also: TDatabase.Params (404), TDatabase.IsSQLBased (403)

14.21.13 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 (404), TDatabase.Directory (402)

14.21.14 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 (397).

See also: TCustomConnection.Connected (397)

14.21.15 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 (402) 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 (402), TDatabase.Params (404)

14.21.16 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 (404)

14.21.17 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 (402), TDatabase.DatabaseName (403)

14.22 TDataLink

14.22.1 Description

`TDataLink` is used by GUI controls or datasets in a master-detail relationship to handle data events coming from a TDatasource (449) 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 (409) 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 (409), TDatasource (449), TDetailDatalink (460), TMasterDataLink (522)

14.22.2 Method overview

Page	Method	Description
405	Create	Initialize a new instance of `TDataLink`.
405	Destroy	Remove an instance of `TDatalink` from memory.
405	Edit	Set the dataset in edit mode, if possible.
406	ExecuteAction	Execute action.
406	UpdateAction	Update handler for actions.
406	UpdateRecord	Called when the data in the dataset must be updated.

14.22.3 Property overview

Page	Properties	Access	Description
406	Active	r	Is the link active.
407	ActiveRecord	rw	Currently active record.
407	BOF	r	Is the dataset at the first record.
407	BufferCount	rw	Set to the number of record buffers this datalink needs.
408	DataSet	r	Dataset this datalink is connected to.
408	DataSource	rw	Datasource this datalink is connected to.
408	DataSourceFixed	rw	Can the datasource be changed.
408	Editing	r	Is the dataset in edit mode.
409	Eof	r	
409	ReadOnly	rw	Is the link readonly.
409	RecordCount	r	Number of records in the buffer of the dataset.

14.22.4 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 (405)

14.22.5 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 (405)

14.22.6 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 (450) method of the DataSource (408) to which the datalink instance is coupled. If the TDatasource.AutoEdit (451) 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 (449), TDatalink.Datasource (408), TDatasource.Edit (450), TDatasource.AutoEdit (451)

14.22.7 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 (404) notification (which could result from writing data to the dataset) will be blocked.

See also: TDatalink.RecordChange (404)

14.22.8 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 (406)

14.22.9 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 (406)

14.22.10 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 (439) whenever the DataSource (408) property is set.

See also: TDatalink.Datasource (408), TDatalink.ReadOnly (409), TDataset.Active (439)

14.22.11 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 (407), TDatalink.EOF (409)

14.22.12 TDataLink.BOF

Synopsis: Is the dataset at the first record.

Declaration: `Property BOF : Boolean`

Visibility: public

Access: Read

Description: `BOF` returns TDataset.BOF (431) if the dataset is available, `True` otherwise.

See also: TDatalink.EOF (409), TDataset.BOF (431)

14.22.13 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 (415), TDatalink.ActiveRecord (407)

14.22.14 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 (408), TDataset (409)

14.22.15 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 (449) instance to get access to the dataset it is connected to. A datalink never points directly to a TDataset (409) 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 (409), TDatasource (449)

14.22.16 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 (408) 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 (408)

14.22.17 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 (405) 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 (405), TDataset.Edit (420)

14.22.18 TDataLink.Eof

Synopsis:

Declaration: `Property Eof : Boolean`

Visibility: public

Access: Read

Description: `EOF` returns TDataset.EOF (433) if the dataset is available, `True` otherwise.

See also: TDatalink.BOF (407), TDataset.EOF (433)

14.22.19 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 (405) 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 (406) property can be used to disable delivery of events to the datalink.

See also: TDatalink.Active (406), TDatalink.edit (405)

14.22.20 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 (407) property: `RecordCount` i s always less than `Buffercount`.

See also: TDatalink.BufferCount (407)

14.23 TDataSet

14.23.1 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 (428) 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 (439) property). Likewise, the Close (418) 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 (427), Prev (409), First (423) and Last (426) methods, it is possible to navigate through the records. The EOF (433) property will be True if the last row has been reached. Likewise, the BOF (431) 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 (436) will be zero.

The Append (416) and Insert (425) methods can be used to insert new records to the set of records. The TDataset.Delete (419) statement is used to delete the current record, and the Edit (420) 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 (428) method, or can be undone using the Cancel (417) method.

The data in the various fields properties is available through the Fields (437) 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 (462)

14.23.2 Method overview

Page	Method	Description
415	ActiveBuffer	Currently active memory buffer.
416	Append	Append a new record to the data.
416	AppendRecord	Append a new record to the dataset and fill with data.
417	BookmarkValid	Test whether `ABookMark` is a valid bookmark.
417	Cancel	Cancel the current editing operation.
417	CheckBrowseMode	Check whether the dataset is in browse mode.
417	ClearFields	Clear the values of all fields.
418	Close	Close the dataset.
418	CompareBookmarks	Compare two bookmarks.
418	ControlsDisabled	Check whether the controls are disabled.
415	Create	Create a new `TDataset` instance.
419	CreateBlobStream	Create blob stream.
419	CursorPosChanged	Indicate a change in cursor position.
419	DataConvert	Convert data from/to native format.
419	Delete	Delete the current record.
415	Destroy	Free a `TDataset` instance.
420	DisableControls	Disable event propagation of controls.
420	Edit	Set the dataset in editing mode.
421	EnableControls	Enable event propagation of controls.
421	FieldByName	Search a field by name.
421	FindField	Find a field by name.
422	FindFirst	Find the first active record (deprecated).
422	FindLast	Find the last active record (deprecated).
422	FindNext	Find the next active record (deprecated).
422	FindPrior	Find the previous active record (deprecated).
423	First	Position the dataset on the first record.
423	FreeBookmark	Free a bookmark obtained with `GetBookmark` (deprecated).
423	GetBookmark	Get a bookmark pointer (deprecated).
424	GetCurrentRecord	Copy the data for the current record in a memory buffer.
415	GetFieldData	Get the data for a field.
424	GetFieldList	Return field instances in a list.
424	GetFieldNames	Return a list of all available field names.
424	GotoBookmark	Jump to bookmark.
425	Insert	Insert a new record at the current position.
425	InsertRecord	Insert a new record with given values.
425	IsEmpty	Check if the dataset contains no data.
425	IsLinkedTo	Check whether a datasource is linked to the dataset.
426	IsSequenced	Is the data sequenced.
426	Last	Navigate forward to the last record.
426	Locate	Locate a record based on some key values.
427	Lookup	Search for a record and return matching values.
427	MoveBy	Move the cursor position.
427	Next	Go to the next record in the dataset.
428	Open	Activate the dataset: Fetch data into memory.
428	Post	Post pending edits to the database.
429	Prior	Go to the previous record.
429	Refresh	Refresh the records in the dataset.
429	Resync	Resynchronize the data buffer.
416	SetFieldData	Store the data for a field.
429	SetFields	Set a number of field values at once.
430	Translate	Transliterate a buffer.
430	UpdateCursorPos	Update cursor position.
430	UpdateRecord	Indicate that the record contents have changed.
431	UpdateStatus	Get the update status for the current record.

14.23.3 Property overview

Page	Properties	Access	Description
439	Active	rw	Is the dataset open or closed.
443	AfterCancel	rw	Event triggered after a Cancel operation.
440	AfterClose	rw	Event triggered after the dataset is closed.
444	AfterDelete	rw	Event triggered after a successful Delete operation.
442	AfterEdit	rw	Event triggered after the dataset is put in edit mode.
441	AfterInsert	rw	Event triggered after the dataset is put in insert mode.
440	AfterOpen	rw	Event triggered after the dataset is opened.
442	AfterPost	rw	Event called after changes have been posted to the underlying database.
445	AfterRefresh	rw	Event triggered after the data has been refreshed.
444	AfterScroll	rw	Event triggered after the cursor has changed position.
439	AutoCalcFields	rw	How often should the value of calculated fields be calculated.
443	BeforeCancel	rw	Event triggered before a Cancel operation.
440	BeforeClose	rw	Event triggered before the dataset is closed.
443	BeforeDelete	rw	Event triggered before a Delete operation.
441	BeforeEdit	rw	Event triggered before the dataset is put in edit mode.
441	BeforeInsert	rw	Event triggered before the dataset is put in insert mode.
440	BeforeOpen	rw	Event triggered before the dataset is opened.
442	BeforePost	rw	Event called before changes are posted to the underlying database.
445	BeforeRefresh	rw	Event triggered before the data is refreshed.
444	BeforeScroll	rw	Event triggered before the cursor changes position.
431	BlockReadSize	rw	Number of records to read.
431	BOF	r	Is the cursor at the beginning of the data (on the first record).
431	Bookmark	rw	Get or set the current cursor position.
432	CanModify	r	Can the data in the dataset be modified.
433	DataSource	r	Datasource this dataset is connected to.
433	DefaultFields	r	Is the dataset using persistent fields or not.
433	EOF	r	Indicates whether the last record has been reached.
434	FieldCount	r	Number of fields.
434	FieldDefs	rw	Definitions of available fields in the underlying database.
437	Fields	r	Indexed access to the fields of the dataset.
438	FieldValues	rw	Access to field values based on the field names.
438	Filter	rw	Filter to apply to the data in memory.
438	Filtered	rw	Is the filter active or not.
439	FilterOptions	rw	Options to apply when filtering.
435	Found	r	Check success of one of the `Find` methods.
435	IsUniDirectional	r	Is the dataset unidirectional (i.e. forward scrolling only).
435	Modified	r	Was the current record modified ?
445	OnCalcFields	rw	Event triggered when values for calculated fields must be computed.
446	OnDeleteError	rw	Event triggered when a delete operation fails.
446	OnEditError	rw	Event triggered when an edit operation fails.
447	OnFilterRecord	rw	Event triggered to filter records.
447	OnNewRecord	rw	Event triggered when a new record is created.
447	OnPostError	rw	Event triggered when a post operation fails.
436	RecNo	rw	Current record number.
436	RecordCount	r	Number of records in the dataset.
436	RecordSize	r	Size of the record in memory.
437	SparseArrays	rw	
437	State	r	Current operational state of the dataset.

14.23.4 TDataSet.Create

Synopsis: Create a new `TDataset` instance.

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

Visibility: public

Description: `Create` initializes a new TDataset (409) 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 (415)

14.23.5 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 (418), TDataset.Create (415)

14.23.6 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.

14.23.7 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 (476)

14.23.8 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 (476)

14.23.9 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 (425) 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 (425), TDataset.Edit (420)

14.23.10 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 (429)) and attempts to post the record using TDataset.Post (428). 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 (416), TDataset.SetFields (429), TDataset.Post (428)

14.23.11 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 (423), TDataset.SetBookmark (409), TDataset.FreeBookmark (423), TDataset.BookmarkAvailable (409)

14.23.12 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 (443) and TDataset.AfterCancel (443) events. If the dataset was sin insert mode, then the TDataset.BeforeScroll (444) and TDataset.AfterScroll (444) 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 (437), TDataset.Append (416), TDataset.Insert (425), TDataset.Edit (420)

14.23.13 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 (371) exception is raised. If it is active, but in an edit state, then TDataset.UpdateRecord (430) is called, and if the TDataset.Modified (435) property is true, a TDataset.Post (428) is performed, else TDataset.Cancel (417) is called.

See also: TDataset.State (437), TDataset.Post (428), TDataset.Cancel (417), TDataset.UpdateRecord (430), TDataset.Modified (435)

14.23.14 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 (371) exception will be raised.

See also: TDataset.State (437), TField.Clear (466)

14.23.15 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 (440) and TDataset.AfterClose (440) 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 (439), TDataset.Open (428)

14.23.16 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 (420) call can be used to disable sending of data events to the controls. The sending can be re-enabled with TDataset.EnableControls (421). 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 (420), TDataset.EnableControls (421)

14.23.17 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 (417), TDataset.GetBookmark (423), TDataset.SetBookmark (409)

14.23.18 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 (386) and TBlobField.SaveToStream (387) 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 (386), TBlobField.SaveToStream (387)

14.23.19 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).

14.23.20 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 (415), TDataset.SetFieldData (416)

14.23.21 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 (443), TDataset.BeforeScroll (444), TDataset.AfterDelete (444) and TDataset.AfterScroll (444) 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 (371) exception will be raised.

See also: TDataset.Cancel (417), TDataset.BeforeDelete (443), TDataset.BeforeScroll (444), TDataset.AfterDelete (444), TDataset.AfterScroll (444)

14.23.22 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 (421). 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 (418).

See also: TDataset.EnableControls (421), TDataset.ControlsDisabled (418)

14.23.23 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 (441) and TDataset.AfterEdit (442) 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 (416) instead.

Errors: If the dataset is read-only or not opened, then an EDatabaseError (371) exception will be raised.

See also: TDataset.State (437), TDataset.EOF (433), TDataset.BOF (431), TDataset.Append (416), TDataset.BeforeEdit (441), TDataset.AfterEdit (442)

14.23.24 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 (420) 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 (418).

See also: TDataset.DisableControls (420), TDataset.ControlsDisabled (418)

14.23.25 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 (499): 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 (371) exception will be raised.

See also: TFields.FieldByname (499), TDataset.FindField (421)

14.23.26 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 (499): 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 (421), TFields.FindField (499)

14.23.27 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 (423) instead.

See also: TDataset.First (423), TDataset.FindLast (422), TDataset.FindNext (422), TDataset.FindPrior (422)

14.23.28 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 (426) instead.

See also: TDataset.Last (426), TDataset.FindFirst (422), TDataset.FindNext (422), TDataset.FindPrior (422)

14.23.29 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 (427) instead.

See also: TDataset.Next (427), TDataset.FindFirst (422), TDataset.FindLast (422), TDataset.FindPrior (422)

14.23.30 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 (429) instead.

See also: TDataset.Prior (429), TDataset.FindFirst (422), TDataset.FindLast (422), TDataset.FindPrior (422)

14.23.31 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 (444) and TDataset.AfterScroll (444) events. After the action is completed, the TDataset.BOF (431) property will be `True`.

Errors: If the dataset is unidirectional or is closed, an EDatabaseError (371) exception will be raised.

See also: TDataset.Prior (429), TDataset.Last (426), TDataset.Next (427), TDataset.BOF (431), TDataset.BeforeScroll (444), TDataset.AfterScroll (444)

14.23.32 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 (423). It should not be used on bookmarks obtained with the TDataset.Bookmark (431) 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 (423), TDataset.Bookmark (431)

14.23.33 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 (409) 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 (423) call. The bookmark will be `Nil` if the dataset is empty or not active.

This call is deprecated. Use the TDataset.Bookmark (431) property instead to get a bookmark.

See also: TDataset.SetBookmark (409), TDataset.FreeBookmark (423), TDataset.Bookmark (431)

14.23.34 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 (415)

14.23.35 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 (371) exception will be raised.

See also: TDataset.GetFieldNames (424), TDataset.FieldByName (421), TDataset.FindField (421)

14.23.36 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 (409), TDataset.FieldByName (421), TDataset.FindField (421)

14.23.37 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 (423) function.

This function is deprecated, use the TDataset.Bookmark (431) property instead.

Errors: if `ABookmark` does not contain a valid bookmark, then an exception may be raised.

See also: TDataset.Bookmark (431), TDataset.GetBookmark (423), TDataset.FreeBookmark (423)

14.23.38 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 (441), BeforeScroll (444), OnNewRecord (447), AfterInsert (441) and AfterScroll (444) 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 (428) must be called.

Errors: If the dataset is read-only, calling `Insert` will result in an EDatabaseError (371).

See also: BeforeInsert (441), BeforeScroll (444), OnNewRecord (447), AfterInsert (441), AfterScroll (444), TDataset.Post (428), TDataset.Append (416)

14.23.39 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 (425), TDataset.SetFieldValues (409)

14.23.40 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 (433) and TDataset.BOF (431) are both `True`, and the dataset is not in insert mode.

See also: TDataset.EOF (433), TDataset.BOF (431), TDataset.State (437)

14.23.41 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" (451) points to the current dataset instance, or indirectly.

See also: TDatasource.Dataset (451)

14.23.42 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 (436) 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 (436)

14.23.43 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 (427). After a call to `Last`, the TDataset.EOF (433) property will be `True`.

Calling this method will trigger the TDataset.BeforeScroll (444) and TDataset.AfterScroll (444) events.

See also: TDataset.First (423), TDataset.Next (427), TDataset.EOF (433), TDataset.BeforeScroll (444), TDataset.AfterScroll (444)

14.23.44 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 (363).

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 (409), TDataset.Lookup (427), TLocateOption (363)

14.23.45 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 (426) 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 (371) exception will be raised.

See also: TDataset.Locate (426)

14.23.46 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 (444) and TDataset.AfterScroll (444) 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 (371) exception on unidirectional datasets.

See also: TDataset.RecNo (436), TDataset.BeforeScroll (444), TDataset.AfterScroll (444)

14.23.47 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 (444) and TDataset.AfterScroll (444) events. If the dataset is located on the last known record (EOF (433) 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 (371) exception.

See also: TDataset.MoveBy (427), TDataset.Prior (429), TDataset.Last (426), TDataset.BeforeScroll (444), TDataset.AfterScroll (444), TDataset.EOF (433)

14.23.48 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 (431)is set to `True`
4. Internal buffers are allocated and filled with data
5. If the dataset is empty, EOF (433) is set to true
6. State (437) is set to `dsBrowse`
7. The AfterOpen (440) 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 (439), TDataset.State (437), TDataset.BOF (431), TDataset.EOF (433), TDataset.BeforeOpen (440), TDataset.AfterOpen (440)

14.23.49 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 (425), Append (416) or TDataset.Edit (420) 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 (442), AfterPost (442) events. After the call to `Past`, the State (437) 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 (352)) will result in an EdatabaseError (371) exception. If an exception occurs during the save operation, the OnPostError (447) event is triggered to handle the error.

See also: Insert (425), Append (416), Edit (420), OnPostError (447), BeforePost (442), AfterPost (442), State (437)

14.23.50 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 (444) and TDataset.AfterScroll (444) events. If the dataset is located on the first record, (BOF (431) 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 (371) exception.

See also: TDataset.MoveBy (427), TDataset.Next (427), TDataset.First (423), TDataset.BeforeScroll (444), TDataset.AfterScroll (444), TDataset.BOF (431)

14.23.51 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 (445) and AfterRefresh (445) events.

Errors: Refreshing may fail if the underlying dataset descendent does not support it.

See also: TDataset.Close (418), TDataset.Open (428), BeforeRefresh (445), AfterRefresh (445)

14.23.52 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 (429) should be used. The `Resync` parameter indicates how the buffers should be refreshed.

See also: TDataset.Refresh (429)

14.23.53 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 (437) property, and works it's way down the array.

Errors: If the dataset is not in edit mode, then an EDatabaseError (371) exception will be raised. If there are more values than fields, an `EListError` exception will be raised.

See also: TDataset.Fields (437)

14.23.54 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 (555) 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 (555)

14.23.55 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 (429)

14.23.56 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 (409)

14.23.57 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.

14.23.58 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 (351), DisableControls (351)

14.23.59 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 (423). As soon as TDataset.Next (427) is called, `BOF` will no longer be true.

See also: TDataset.EOF (433), TDataset.Next (427), TDataset.First (423)

14.23.60 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 (420) and TDataset.EnableControls (421) 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 (420), TDataset.EnableControls (421)

14.23.61 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 (425), TDataset.Append (416), TDataset.Delete (419), Tdataset.Edit (420)

14.23.62 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 (449)

14.23.63 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 (434) 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 (434), TDataset.Fields (437), TField (462)

14.23.64 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 (427) method is called.
2. The TDataset.Last (426) method is called (which is equivalent to moving to the last record and calling TDataset.Next (427)).
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 (431), TDataset.Next (427), TDataset.Last (426), TDataset.IsEmpty (425)

14.23.65 TDataSet.FieldCount

Synopsis: Number of fields.

Declaration: `Property FieldCount : LongInt`

Visibility: public

Access: Read

Description: `FieldCount` is the same as Fields.Count (501), i.e. the number of fields. For a dataset with persistent fields (when DefaultFields (433) 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 (497)

14.23.66 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 (433) 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 (428), TDataset.DefaultFields (433), TDataset.Fields (437)

14.23.67 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 (422),TDataset.FindLast (422),TDataset.FindNext (422) or TDataset.FindPrior (422) operations was successful.

See also: TDataset.FindFirst (422), TDataset.FindLast (422), TDataset.FindNext (422), TDataset.FindPrior (422)

14.23.68 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 (420) or Tdataset.Insert (425). It becomes `True` if a value was written to one of the fields of the dataset.

See also: Tdataset.Edit (420), TDataset.Insert (425), TDataset.Append (416), TDataset.Cancel (417), TDataset.Post (428)

14.23.69 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 (429) or TDataset.Last (426)), random positioning of the cursor, editing or filtering will result in an EDatabaseError (371). 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 (429), TDataset.Next (427)

14.23.70 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 (436)

14.23.71 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 (436)

14.23.72 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 (475), TDataset.RecordCount (436), TDataset.RecNo (436)

14.23.73 TDataSet.SparseArrays

Declaration: `Property SparseArrays : Boolean`

Visibility: public

Access: Read,Write

14.23.74 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 (439), TDataset.Edit (420), TDataset.Insert (425), TDataset.Append (416), TDataset.Post (428), TDataset.Cancel (417)

14.23.75 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 (497) 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 (494), TField (462)

14.23.76 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 (499)

14.23.77 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 (438), TDataset.FilterOptions (439)

14.23.78 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 (438) 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 (438), TDataset.FilterOptions (439)

14.23.79 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 (438), 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 (438), TDataset.FilterOptions (439)

14.23.80 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 (437) is then `dsInactive`). Setting the `Active` property to `True` is equivalent to calling TDataset.Open (428), setting it to `False` is equivalent to calling TDataset.Close (418)

See also: TDataset.State (437), TDataset.Open (428), TDataset.Close (418)

14.23.81 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 (445) event.

See also: TField.FieldKind (482), TDataset.OnCalcFields (445)

14.23.82 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 (440), TDataset.State (437)

14.23.83 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 (444) 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 (440), TDataset.State (437), TDataset.AfterScroll (444)

14.23.84 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 (440), TDataset.State (437)

14.23.85 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 (440), TDataset.State (437)

14.23.86 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 (416) or TDataset.Insert (425) 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 (441), TDataset.Append (416), TDataset.Insert (425)

14.23.87 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 (444) 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 (441), TDataset.AfterScroll (444), TDataset.Append (416), TDataset.Insert (425)

14.23.88 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 (420) 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 (442), TDataset.Edit (420), TDataset.State (437)

14.23.89 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 (441), TDataset.Edit (420), TDataset.State (437)

14.23.90 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 (428) 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 (428), TDataset.AfterPost (442), TDataset.State (437)

14.23.91 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 (428) 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 (447) event is triggered instead.

See also: TDataset.BeforePost (442), TDataset.Post (428), TDataset.State (437), TDataset.OnPostError (447)

14.23.92 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 (417) 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 (443), TDataset.Cancel (417), TDataset.State (437)

14.23.93 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 (417) operation was successfully completed, and the dataset is again in `dsBrowse` state.

See also: TDataset.BeforeCancel (443), TDataset.Cancel (417), TDataset.State (437)

14.23.94 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 (419) 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 (444) event. If the dataset was in insert mode when the `Delete` method was called, then the event will not be called, as TDataset.Cancel (417) is called instead.

See also: TDataset.AfterDelete (444), TDataset.Delete (419), TDataset.BeforeScroll (444), TDataset.Cancel (417), TDataset.State (437)

14.23.95 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 (419) operation, when the dataset is again in `dsBrowse` state. The event is followed by a TDataset.AfterScroll (444) event.

See also: TDataset.BeforeDelete (443), TDataset.Delete (419), TDataset.AfterScroll (444), TDataset.State (437)

14.23.96 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 (427), TDataset.Prior (429), TDataset.First (423), TDataset.Last (426), but also with two of the editing operations:TDataset.Insert (425) and TDataset.Delete (419). Raising an exception in this event handler aborts the operation in progress.

See also: TDataset.AfterScroll (444), TDataset.Next (427), TDataset.Prior (429), TDataset.First (423), TDataset.Last (426), TDataset.Insert (425), TDataset.Delete (419)

14.23.97 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 (427), TDataset.Prior (429), TDataset.First (423), TDataset.Last (426), but also with two of the editing operations:TDataset.Insert (425) and TDataset.Delete (419) 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 (444), TDataset.Next (427), TDataset.Prior (429), TDataset.First (423), TDataset.Last (426), TDataset.Insert (425), TDataset.Delete (419), TDataset.Open (428)

14.23.98 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 (429) 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 (429), TDataset.AfterRefresh (445), TDataset.State (437)

14.23.99 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 (429) 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 (429), TDataset.State (437), TDataset.BeforeRefresh (445)

14.23.100 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 (439) 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 (439), TField.Kind (462)

14.23.101 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 (419) 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 (356) event handler type.

See also: TDatasetErrorEvent (356), TDataset.Delete (419), TDataset.OnEditError (446), TDataset.OnPostError (447)

14.23.102 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 (420) 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 (356) event handler type.

See also: TDatasetErrorEvent (356), TDataset.Edit (420), TDataset.OnDeleteError (446), TDataset.OnPostError (447)

14.23.103 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 (438) 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 (438), TDataset.Filtered (438), TDataset.state (437)

14.23.104 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 (416) or TDataset.Insert (425) 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 (437) is again `dsBrowse`).

See also: TDataset.Append (416), TDataset.Insert (425), TDataset.State (437)

14.23.105 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 (428) 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 (356) event handler type.

See also: TDatasetErrorEvent (356), TDataset.Post (428), TDataset.OnDeleteError (446), TDataset.OnEditError (446)

14.24 TDataSetEnumerator

14.24.1 Description

`TDataSetEnumerator` is an enumerator for the records in a dataset. It returns the TDataset.Fields (437) 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 (409), TFields (497), TFieldsEnumerator (501)

14.24.2 Method overview

Page	Method	Description
448	Create	Create a new instance of the dataset enumerator.
448	MoveNext	Attempts to navigate to the next record.

14.24.3 Property overview

Page	Properties	Access	Description
449	Current	r	Current record.

14.24.4 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 (409), TDataset.First (423), TFieldsEnumerator (501)

14.24.5 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 (427), TDataset.EOF (433)

14.24.6 TDataSetEnumerator.Current

Synopsis: Current record.

Declaration: `Property Current : TFields`

Visibility: public

Access: Read

Description: `Current` always returns TDataset.Fields (437).

See also: TDataset.Fields (437)

14.25 TDataSource

14.25.1 Description

`TDatasource` is a mediating component: it handles communication between any DB-Aware component (often edit controls on a form) and a TDataset (409) instance. Any database aware component should never communicate with a dataset directly. Instead, it should communicate with a TDatasource (449) 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 (409), TDatalink (404)

14.25.2 Method overview

Page	Method	Description
450	Create	Create a new instance of `TDatasource`.
450	Destroy	Remove a `TDatasource` instance from memory.
450	Edit	Put the dataset in edit mode, if needed.
450	IsLinkedTo	Check if a dataset is linked to a certain dataset.

14.25.3 Property overview

Page	Properties	Access	Description
451	AutoEdit	rw	Should the dataset be put in edit mode automatically.
451	DataSet	rw	Dataset this datasource is connected to.
452	Enabled	rw	Enable or disable sending of events.
452	OnDataChange	rw	Called whenever data changes in the current record.
452	OnStateChange	rw	Called whenever the state of the dataset changes.
453	OnUpdateData	rw	Called whenever the data in the dataset must be updated.
451	State	r	State of the dataset.

14.25.4 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 (450)

14.25.5 TDataSource.Destroy

Synopsis: Remove a `TDatasource` instance from memory.

Declaration: `destructor Destroy; Override`

Visibility: public

Description: `Destroy` notifies all TDataLink (404) 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 (450), TDatalink (404)

14.25.6 TDataSource.Edit

Synopsis: Put the dataset in edit mode, if needed.

Declaration: `procedure Edit`

Visibility: public

Description: `Edit` will check AutoEdit (451): if it is `True`, then it puts the Dataset (451) 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 (420) directly.

Errors: An EDatabaseError (371) 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 (451), TDataset.Edit (420), TDataset.State (437)

14.25.7 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 (451) 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 (451)

14.25.8 TDataSource.State

Synopsis: State of the dataset.

Declaration: `Property State : TDataSetState`

Visibility: public

Access: Read

Description: `State` contains the State (437) 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 (437) directly, to take into account the effect of the Enabled (452) property.

See also: TDataset.State (437), TDatasource.Enabled (452)

14.25.9 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 (450) 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 (450), TDataset.Edit (420)

14.25.10 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 (409) 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 (409), TDatasource.Enabled (452)

14.25.11 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 (409), TDatasource.Dataset (451), TDatasource.AutoEdit (451)

14.25.12 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 (437) 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 (453), TDatasource.OnStateChange (452), TDataset.State (437), TDatasource.Enabled (452)

14.25.13 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 (453), TDatasource.OnStateChange (452), TDataset.AfterScroll (444), TField.OnChange (487), TDatasource.Enabled (452)

14.25.14 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 (428) 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 (452), TDatasource.OnStateChange (452), TDataset.Post (428)

14.26 TDateField

14.26.1 Description

`TDateField` is the class used when a dataset must manage data of type date. (TField.DataType (475) equals `ftDate`). It initializes some of the properties of the TField (462) 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 (409), TField (462), TDateTimeField (453), TTimeField (556)

14.26.2 Method overview

Page	Method	Description
453	Create	Create a new instance of a `TDateField` class.

14.26.3 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 (462) properties to configure the instance for working with date values.

See also: TField (462)

14.27 TDateTimeField

14.27.1 Description

`TDateTimeField` is the class used when a dataset must manage data of type datetime. (TField.DataType (475) equals `ftDateTime`). It also serves as base class for the TDateField (453) or TTimeField (556) classes. It overrides some of the properties and methods of the TField (462) 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 (409), TField (462), TDateField (453), TTimeField (556)

14.27.2 Method overview

Page	Method	Description
454	Create	Create a new instance of a `TDateTimeField` class.

14.27.3 Property overview

Page	Properties	Access	Description
454	DisplayFormat	rw	Formatting string for textual representation of the field.
455	EditMask		Specify an edit mask for an edit control.
454	Value	rw	Contents of the field as a TDateTime value.

14.27.4 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 (462) properties to configure the instance for working with date/time values.

See also: TField (462)

14.27.5 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 (479) by `TDateTimeField` as a `TDateTime` value. It returns the same value as the TField.AsDateTime (470) property.

See also: TField.AsDateTime (470), TField.Value (479)

14.27.6 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 (476) property to format the value with the DateTimeToString (**??**)function.

See also: DateTimeToString (**??**), FormatDateTime (**??**), TField.DisplayText (476)

14.27.7 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 (476)

14.28 TDBDataset

14.28.1 Description

`TDBDataset` is a `TDataset` descendent which introduces the concept of a database: a central component (TDatabase (400)) which represents a connection to a database. This central component is exposed in the TDBDataset.Database (456) 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 (456) 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 (400), TDBTransaction (456)

14.28.2 Method overview

Page	Method	Description
456	destroy	Remove the `TDBDataset` instance from memory.

14.28.3 Property overview

Page	Properties	Access	Description
456	DataBase	rw	Database this dataset is connected to.
456	Transaction	rw	Transaction in which this dataset is running.

14.28.4 TDBDataset.destroy

Synopsis: Remove the `TDBDataset` instance from memory.

Declaration: `destructor destroy; Override`

Visibility: public

Description: `Destroy` will disconnect the `TDBDataset` from its Database (456) and Transaction (456). After this it calls the inherited destructor.

See also: TDBDataset.Database (456), TDatabase (400)

14.28.5 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 (400) 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 (371) exception will be raised.

See also: TDatabase (400), TDBDataset.Transaction (456)

14.28.6 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 (456) 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 (456), TDBDataset.Database (456)

14.29 TDBTransaction

14.29.1 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 (400) instance, and to keep track of `TDataset` instances which are connected to the transaction.

See also: TDatabase (400), TDataset (409)

14.29.2 Method overview

Page	Method	Description
457	CloseDataSets	Close all connected datasets.
457	Create	Transaction property.
457	Destroy	Remove a `TDBTransaction` instance from memory.

14.29.3 Property overview

Page	Properties	Access	Description
458	Active	rw	Is the transaction active or not.
458	DataBase	rw	Database this transaction is connected to.

14.29.4 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 (457)

14.29.5 TDBTransaction.Destroy

Synopsis: Remove a `TDBTransaction` instance from memory.

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

Visibility: public

Description: `Destroy` first disconnects all connected TDBDataset (455) instances and then cleans up the resources allocated in the Create (457) constructor. After that it calls the inherited destructor.

See also: TDBTransaction.Create (457)

14.29.6 TDBTransaction.CloseDataSets

Synopsis: Close all connected datasets.

Declaration:
```
procedure CloseDataSets
procedure CloseDataSets(InCommit: Boolean)
```

Visibility: public

Description: `CloseDatasets` closes all connected datasets (All TDBDataset (455) instances whose Transaction (456) property points to this `TDBTransaction` instance).

See also: TDBDataset (455), TDBDataset.Transaction (456)

14.29.7 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 (371) exception.

See also: TDBTransaction.Active (458), TDatabase (400)

14.29.8 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 (456). Reversely, setting `Active` to `True` will call `StartTransaction`, setting it to `False` will call TDBTransaction.EndTransaction (456).

See also: TDBTransaction.StartTransaction (456), TDBTransaction.EndTransaction (456)

14.30 TDefCollection

14.30.1 Description

`TDefCollection` is a parent class for the TFieldDefs (494) and TIndexDefs (512) collections: It holds a set of named definitions on behalf of a TDataset (409) component. To this end, it introduces a dataset (460) property, and a mechanism to notify the dataset of any updates in the collection. It is supposed to hold items of class TNamedItem (527), so the TDefCollection.Find (459) method can find items by named.

14.30.2 Method overview

Page	Method	Description
459	create	Instantiate a new `TDefCollection` instance.
459	Find	Find an item by name.
459	GetItemNames	Return a list of all names in the collection.
459	IndexOf	Find location of item by name.

14.30.3 Property overview

Page	Properties	Access	Description
460	Dataset	r	Dataset this collection manages definitions for.
460	Updated	rw	Has one of the items been changed.

14.30.4 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 (409), TNamedItem (527)

14.30.5 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 (527), TDefCollection.IndexOf (459)

14.30.6 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 (527)

14.30.7 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 (459), TNamedItem.Name (527)

14.30.8 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.

14.30.9 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.

14.31 TDetailDataLink

14.31.1 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 (409), TDatasource (449)

14.31.2 Property overview

Page	Properties	Access	Description
460	DetailDataSet	r	Detail dataset in Master-detail relation.

14.31.3 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 (522). The master dataset is available through the regular TDataLink.DataSource (408) property.

See also: TDataset (409), TMasterDatalink (522), TDataLink.DataSource (408)

14.32 TExtendedField

14.32.1 Method overview

Page	Method	Description
461	CheckRange	
461	Create	

14.32.2 Property overview

Page	Properties	Access	Description
461	Currency	rw	
461	MaxValue	rw	
462	MinValue	rw	
462	Precision	rw	
461	Value	rw	

14.32.3 TExtendedField.Create

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

Visibility: public

14.32.4 TExtendedField.CheckRange

Declaration: `function CheckRange(AValue: Extended) : Boolean`

Visibility: public

14.32.5 TExtendedField.Value

Declaration: `Property Value : Extended`

Visibility: public

Access: Read,Write

14.32.6 TExtendedField.Currency

Declaration: `Property Currency : Boolean`

Visibility: published

Access: Read,Write

14.32.7 TExtendedField.MaxValue

Declaration: `Property MaxValue : Extended`

Visibility: published

Access: Read,Write

14.32.8 TExtendedField.MinValue

Declaration: `Property MinValue : Extended`

Visibility: published

Access: Read,Write

14.32.9 TExtendedField.Precision

Declaration: `Property Precision : LongInt`

Visibility: published

Access: Read,Write

14.33 TField

14.33.1 Description

`TField` is an abstract class that defines access methods for a field in a record, controlled by a TDataset (409) 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 (409), TFieldDef (488), TFields (497)

14.33.2 Method overview

Page	Method	Description
465	Assign	Copy properties from one TField instance to another.
465	AssignValue	Assign value of a variant record to the field.
466	Clear	Clear the field contents.
465	Create	Create a new `TField` instance.
465	Destroy	Destroy the `TField` instance.
466	FocusControl	Set focus to the first control connected to this field.
466	GetData	Get the data from this field.
467	IsBlob	Is the field a BLOB field (untyped data of indeterminate size).
467	IsValidChar	Check whether a character is valid input for the field.
467	RefreshLookupList	Refresh the lookup list.
467	SetData	Save the field data.
468	SetFieldType	Set the field data type.
468	Validate	Validate the data buffer.

14.33.3 Property overview

Page	Properties	Access	Description
480	Alignment	rw	Alignment for this field.
472	AsAnsiString	rw	Return field contents as an ANSI string.
468	AsBCD	rw	Access the field's contents as a BCD (Binary coded Decimal).
469	AsBoolean	rw	Access the field's contents as a Boolean value.
469	AsBytes	rw	Retrieve the contents of the field as an array of bytes.
469	AsCurrency	rw	Access the field's contents as a Currency value.
470	AsDateTime	rw	Access the field's contents as a TDateTime value.
470	AsExtended	rw	
470	AsFloat	rw	Access the field's contents as a floating-point (Double) value.
471	AsInteger	rw	Access the field's contents as a 32-bit signed integer (longint) value.
471	AsLargeInt	rw	Access the field's contents as a 64-bit signed integer (longint) value.
470	AsLongint	rw	Access the field's contents as a 32-bit signed integer (longint) value.
471	AsLongWord	rw	Access field contents as 32-bit unsigned integer (longword/cardinal).
472	AsSingle	rw	
472	AsString	rw	Access the field's contents as an AnsiString value.
473	AsUnicodeString	rw	Field contents as a UnicodeString.
473	AsUTF8String	rw	Field contents as a UTF8 String.
474	AsVariant	rw	Access the field's contents as a Variant value.
473	AsWideString	rw	Access the field's contents as a WideString value.
474	AttributeSet	rw	Not used: dictionary information.
474	Calculated	rw	Is the field a calculated field ?
474	CanModify	r	Can the field's contents be modified.
481	ConstraintErrorMessage	rw	Message to display if the `CustomConstraint` constraint is violated.
475	CurValue	r	Current value of the field.
481	CustomConstraint	rw	Custom constraint for the field's value.
475	DataSet	rw	Dataset this field belongs to.
475	DataSize	r	Size of the field's data.
475	DataType	r	The data type of the field.
481	DefaultExpression	rw	Default value for the field.
482	DisplayLabel	rws	Name of the field for display purposes.
476	DisplayName	r	User-readable fieldname.
476	DisplayText	r	Formatted field value.
482	DisplayWidth	rws	Width of the field in characters.
476	EditMask	rw	Specify an edit mask for an edit control.
477	EditMaskPtr	r	Alias for EditMask.
480	FieldDef	r	Fielddef associated with this field.
482	FieldKind	rw	The kind of field.
482	FieldName	rw	Name of the field.
477	FieldNo	r	Number of the field in the record.
483	HasConstraints	r	Does the field have any constraints defined.
483	ImportedConstraint	rw	Constraint for the field value on the level of the underlying database.
483	Index	rw	Index of the field in the list of fields.
477	IsIndexField	r	Is the field an indexed field ?
477	IsNull	r	Is the field empty.
484	KeyFields	rw	Key fields to use when looking up a field value.
485	Lookup	rws	Is the field a lookup field.
484	LookupCache	rw	Should lookup values be cached.
484	LookupDataSet	rw	Dataset with lookup values.
484	LookupKeyFields	rw	Names of fields on which to perform a locate.
480	LookupList	r	List of lookup values.
485	LookupResultField	rw	Name of field to use as lookup value.

14.33.4 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 (428) method will create the necessary `TField` instances, if none had been created in the designer.

See also: TDataset.Open (428)

14.33.5 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 (418)

14.33.6 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 (479)

14.33.7 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 (371) exception.

See also: TField.Assign (465), TField.Value (479)

14.33.8 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 (477) 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 (371) exception.

See also: TField.IsNull (477), TField.Value (479)

14.33.9 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 (421), TDataset.DisableControls (420)

14.33.10 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 (477), TField.SetData (467), TField.Value (479)

14.33.11 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 (385)

14.33.12 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 (479) 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 (479)

14.33.13 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 (484) 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 (484), LookupKeyFields (484), LookupResultField (485)

14.33.14 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 (466), TField.Value (479)

14.33.15 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 (475)

14.33.16 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 (488) event handler, if one is set, to allow the application programmer to program additional checks.

See also: TField.SetData (467), TField.OnValidate (488)

14.33.17 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 (469), TField.Value (479)

14.33.18 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 (479), TField.AsInteger (471)

14.33.19 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 (384)

14.33.20 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 (479), TField.AsFloat (470)

14.33.21 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 (479), TField.AsString (472)

14.33.22 TField.AsExtended

Declaration: `Property AsExtended : Extended`

Visibility: public

Access: Read,Write

14.33.23 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 (479), TField.AsString (472), TField.AsCurrency (469)

14.33.24 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 (471).

See also: TField.Value (479), TField.AsString (472), TField.AsInteger (471)

14.33.25 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 (479), TField.AsInteger (471)

14.33.26 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 (479), TField.AsString (472), TField.AsInteger (471)

14.33.27 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 (479), TField.AsString (472), TField.AsLongint (470), TField.AsInt64 (462)

14.33.28 TField.AsSingle

Declaration: `Property AsSingle : Single`

Visibility: public

Access: Read,Write

14.33.29 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 (479), TField.AsWideString (473)

14.33.30 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 (472), TField.AsUnicodeString (473), TField.AsUTF8String (473), TField.CodePage (462)

14.33.31 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 (462) 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 (472), TField.AsAnsiString (472), TField.AsUTF8String (473), TField.CodePage (462)

14.33.32 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 (462). 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 (472), TField.AsUnicodeString (473), TField.AsAnsi8String (462), TField.CodePage (462)

14.33.33 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 (479), TField.Astring (462)

14.33.34 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 (479), TField.Astring (462)

14.33.35 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.

14.33.36 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 (482) 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 (482)

14.33.37 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 (486), TDataset.CanModify (432)

14.33.38 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 (479)

14.33.39 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 (409), TDataset.Fields (437)

14.33.40 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 (478) property which declares a logical size for datatypes that have a variable size (such as string fields). For BLOB fields, use the TBlobField.BlobSize (387) property to get the size of the field's contents for the current record..

See also: TField.Size (478), TBlobField.BlobSize (387)

14.33.41 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 (482)

14.33.42 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 (482) property, unless assigned another value.

The use of this property is deprecated. Use DisplayLabel (482) instead.

See also: Tfield.FieldName (482)

14.33.43 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 (472) property, which does not take into account any formatting.

See also: TField.AsString (472)

14.33.44 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 (455), TStringField.EditMask (555)

14.33.45 TField.EditMaskPtr

Synopsis: Alias for EditMask.

Declaration: `Property EditMaskPtr : TEditMask`

Visibility: public

Access: Read

Description: `EditMaskPtr` is a read-only alias for the EditMask (476) property. It is not used.

See also: TField.EditMask (476)

14.33.46 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 (483)

14.33.47 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 (409) can change this to `True`.

See also: TField.Calculated (474)

14.33.48 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 (425) is called or Clear (466) is called then `IsNull` will return `True`.

See also: TField.Clear (466), TDataset.Insert (425)

14.33.49 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 (409) does not yet support this.

See also: TField.Value (479), TField.CurValue (475)

14.33.50 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 (477), TField.Index (483), TField.Datasize (475)

14.33.51 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 (475) should be used. For blob fields, the current size of the data is not guaranteed to be present.

See also: DataSize (475)

14.33.52 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 (487) event handler if a handler was specified. For display purposes, the TField.DisplayText (476) 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 (472) property, this will invoke the Tfield.OnSetText (487) event handler, if one is set.

See also: TField.AsString (472), TField.DisplayText (476), TField.Value (479)

14.33.53 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 (467) function.

See also: TField.IsValidChar (467)

14.33.54 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 (477) 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 (477), TField.Text (478), TField.DisplayText (476)

14.33.55 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 (479), TField.CurValue (475), TField.NewValue (478)

14.33.56 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 (484) property is set to `True`. It can be refreshed using the RefreshLookupList (467) method.

See also: TField.RefreshLookupList (467), TField.LookupCache (484)

14.33.57 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 (434) 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 (434)

14.33.58 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 (476)

14.33.59 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 (481) as a message.

This feature is not yet implemented in FPC.

See also: TField.ConstraintErrorMessage (481), TField.ImportedConstraint (483)

14.33.60 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 (481) is violated.

This feature is not yet implemented in FPC.

See also: TField.CustomConstraint (481)

14.33.61 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 (416) or TDataset.Insert (425) 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 (425), TDataset.Append (416), TDataset.CustomConstraint (409)

14.33.62 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 (482) property, unless assigned another value.

See also: TField.FieldName (482)

14.33.63 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 (478) for string fields) but can be modified to obtain a more appropriate value for the field's expected content.

See also: TField.Alignment (480), TField.DisplayText (476)

14.33.64 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.

14.33.65 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 (488), TField.Size (478), TField.DataType (475)

14.33.66 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 (481) or ImportedConstraint (483) properties is set to a non-empty value.

See also: CustomConstraint (481), ImportedConstraint (483)

14.33.67 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 (477) property should be used for that.

See also: TField.FieldNo (477)

14.33.68 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 (409) descendent.

See also: TField.CustomConstraint (481), TDataset (409), TField.ConstraintErrorMessage (481)

14.33.69 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 (484) in the dataset pointed to by the LookupDataset (484) property.

See also: LookupKeyFields (484), LookupDataset (484)

14.33.70 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 (484) in the dataset pointed to by the LookupDataset (484) property. The list of key, value pairs is available through the TField.LookupList (480) property.

See also: LookupKeyFields (484), LookupDataset (484), TField.LookupList (480)

14.33.71 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 (484) property is used as a list of fields to locate a record in this dataset, and the value of the LookupResultField (485) field is then used as the value of the lookup field.

See also: KeyFields (484), LookupKeyFields (484), LookupResultField (485), LookupCache (484)

14.33.72 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 (484) property. These fields will be used when locating a record corresponding to the values in the TField.KeyFields (484) property.

See also: KeyFields (484), LookupDataset (484), LookupResultField (485), LookupCache (484)

14.33.73 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 (484) 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 (484), LookupDataset (484), LookupKeyFields (484), LookupCache (484)

14.33.74 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 (482) equals `fkLookup`, `False` otherwise. Setting the `Lookup` property will switch the `FieldKind` between the `fkLookup` and `fkData`.

See also: TField.FieldKind (482)

14.33.75 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 (409) descendent or the database engine support retrieval of this property. It can be used to automatically create update statements, together with the TField.ProviderFlags (486) property.

See also: TDataset (409), TField.ProviderFlags (486)

14.33.76 TField.ParentField

Declaration: `Property ParentField : TObjectField`

Visibility: published

Access: Read,Write

14.33.77 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 (485)

14.33.78 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 (492)

14.33.79 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 (409) 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 (428), ReadOnly (486), Visible (487)

14.33.80 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 (486), TField.Required (486)

14.33.81 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 (488) 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 (487)

14.33.82 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 (478) or TField.DisplayText (476) 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 (478), TField.DisplayText (476), TField.OnSetText (487), TFieldGetTextEvent (358)

14.33.83 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 (478) 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 (487) handler, if it exists.

See also: TField.Text (478), TField.OnGetText (487), TFieldGetTextEvent (358)

14.33.84 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 (478), TField.OnGetText (487), TField.OnSetText (487), TField.OnChange (487)

14.34 TFieldDef

14.34.1 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 (434), TFieldDefs (494)

14.34.2 Method overview

Page	Method	Description
490	AddChild	
490	Assign	Assign the contents of one TFieldDef instance to another.
489	Create	Constructor for `TFieldDef`.
490	CreateField	Create `TField` instance based on definitions in current `TFieldDef` instance.
489	Destroy	Free the TFieldDef instance.
490	HasChildDefs	

14.34.3 Property overview

Page	Properties	Access	Description
492	Attributes	rw	Additional attributes of the field.
491	CharSize	r	Character size.
493	ChildDefs	rws	
492	Codepage	r	System code page for the values in string and wide string field types.
493	DataType	rw	Data type for the field.
490	FieldClass	r	TField class used for this fielddef.
491	FieldNo	r	Field number.
491	InternalCalcField	rw	Is this a definition of an internally calculated field ?
492	ParentDef	r	
493	Precision	rw	Precision used in BCD (Binary Coded Decimal) fields.
492	Required	rw	Is the field required ?
493	Size	rw	Size of the buffer needed to store the data of the field.

14.34.4 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 (488), datatype (493), size (493), precision (493), FieldNo (491), Required (492) and CodePage (492) property are all set according to the passed arguments.

Errors: If a duplicate name is passed, then an exception will occur.

See also: Name (488), datatype (493), size (493), precision (493), FieldNo (491), Required (492), CodePage (492)

14.34.5 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 (489)

14.34.6 TFieldDef.AddChild

Declaration: `function AddChild : TFieldDef`

Visibility: public

14.34.7 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 (488), then an exception will be raised.

14.34.8 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 (493) what TField (462) descendent it should create, and then returns a newly created instance of this class. It sets the appropriate defaults for the Size (478), FieldName (482), FieldNo (477), Precision (462), ReadOnly (486) and Required (486) 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 (352) 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 (352), TField (462)

14.34.9 TFieldDef.HasChildDefs

Declaration: `function HasChildDefs : Boolean`

Visibility: public

14.34.10 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 (490) 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 (409), CreateField (490), TField (462)

14.34.11 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.

14.34.12 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 (492) 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 (492), TFieldDef.Size (493)

14.34.13 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.

14.34.14 TFieldDef.ParentDef

Declaration: `Property ParentDef : TFieldDef`

Visibility: public

Access: Read

14.34.15 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`.

14.34.16 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 (489), TFieldDef.DataType (493)

14.34.17 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 (492)

14.34.18 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 (490), TFieldDef.CreateField (490)

14.34.19 TFieldDef.ChildDefs

Declaration: `Property ChildDefs : TFieldDefs`

Visibility: published

Access: Read,Write

14.34.20 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 (493)

14.34.21 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 (493), TFieldDef.DataType (493)

14.35 TFieldDefs

14.35.1 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 (488), TDataset (409)

14.35.2 Method overview

Page	Method	Description
494	Add	Add a new field definition to the collection.
495	AddFieldDef	Add new `TFieldDef`.
495	Assign	Copy all items from one dataset to another.
494	Create	Create a new instance of `TFieldDefs`.
495	Find	Find item by name.
496	MakeNameUnique	Create a unique field name starting from a base name.
496	Update	Force update of definitions.

14.35.3 Property overview

Page	Properties	Access	Description
496	HiddenFields	rw	Should field instances be created for hidden fields.
496	Items	rw	Indexed access to the fielddef instances.
497	ParentDef	r	

14.35.4 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 (488), TDataset (409)

14.35.5 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 (495)

14.35.6 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 (494)

14.35.7 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 (490)

14.35.8 TFieldDefs.Find

Synopsis: Find item by name.

Declaration: `function Find(const AName: string) : TFieldDef`

Visibility: public

Description: `Find` simply calls the inherited TDefCollection.Find (459) to find an item with name `AName` and typecasts the result to `TFieldDef`.

See also: TDefCollection.Find (459), TNamedItem.Name (527)

14.35.9 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 (460)

14.35.10 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 (495), TFieldDef.Name (488)

14.35.11 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 (492)

14.35.12 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 (488)

14.35.13 TFieldDefs.ParentDef

Declaration: `Property ParentDef : TFieldDef`

Visibility: public

Access: Read

14.36 TFields

14.36.1 Description

`TFields` mimics a `TCollection` class for the Fields (437) property of TDataset (409) instance. Since TField (462) is a descendent of `TComponent`, it cannot be an item of a collection, and must be managed by another class.

See also: TField (462), TDataset (409), TDataset.Fields (437)

14.36.2 Method overview

Page	Method	Description
498	Add	Add a new field to the list.
498	CheckFieldName	Check field name for duplicate entries.
498	CheckFieldNames	Check a list of field names for duplicate entries.
499	Clear	Clear the list of fields.
497	Create	Create a new instance of `TFields`.
498	Destroy	Free the `TFields` instance.
499	FieldByName	Find a field based on its name.
499	FieldByNumber	Search field based on its fieldnumber.
499	FindField	Find a field based on its name.
500	GetEnumerator	Return an enumerator for the `for..in` construct.
500	GetFieldNames	Get the list of fieldnames.
500	IndexOf	Return the index of a field instance.
500	Remove	Remove an instance from the list.

14.36.3 Property overview

Page	Properties	Access	Description
501	Count	r	Number of fields in the list.
501	Dataset	r	Dataset the fields belong to.
501	Fields	rw	Indexed access to the fields in the list.

14.36.4 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 (501) property, and initializes an internal list object to store the list of fields.

See also: TDataset (409), TFields.Dataset (501), TField (462)

14.36.5 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 (501). After that it cleans up the internal structures and then calls the inherited destructor.

See also: TDataset (409), TField (462), TFields.Clear (499)

14.36.6 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 (462) 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 (462), TFields.Clear (499)

14.36.7 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 (497)). If it does, then an EDatabaseError (371) exception is raised.

See also: TField.FieldName (482), TFields.Find (497)

14.36.8 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 (498).

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 (482), TFields.CheckFieldName (498), TFields.Find (497)

14.36.9 TFields.Clear

Synopsis: Clear the list of fields.

Declaration: `procedure Clear`

Visibility: public

Description: `Clear` removes all TField (462) var instances from the list. All field instances are freed after they have been removed from the list.

See also: TField (462)

14.36.10 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 (482) property matches `Value`. The search is performed case-insensitively. If no field instance is found, then `Nil` is returned.

See also: TFields.FieldByName (499)

14.36.11 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 (482) 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 (462), where `Nil` is returned if no match is found.

See also: TFields.FindField (499), TFields.FieldName (497), Tfields.FieldByNumber (499), TFields.IndexOf (500)

14.36.12 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 (477) property matches the `FieldNo` parameter. If no such field is found, `Nil` is returned.

See also: TFields.FieldByName (499), TFields.FindField (499), TFields.IndexOf (500)

14.36.13 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 (501)

See also: TFieldsEnumerator (501), #rtl.system.IEnumerable (**??**)

14.36.14 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 (482)

14.36.15 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 (499), TFields.FieldByNumber (499), TFields.FindField (499)

14.36.16 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 (499)

14.36.17 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 (501) property are numbered from 0 to `Count-1`.

See also: TFields.fields (501)

14.36.18 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 (497) 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 (497), TField.Dataset (475), TDataset (409)

14.36.19 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 (499)

14.37 TFieldsEnumerator

14.37.1 Description

`TFieldsEnumerator` implements all the methods of `IEnumerator` so a TFields (497) 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 (462), TFields (497), #rtl.system.IEnumerator (**??**)

14.37.2 Method overview

Page	Method	Description
502	Create	Create a new instance of `TFieldsEnumerator`.
502	MoveNext	Move the current field to the next field in the collection.

14.37.3 Property overview

Page	Properties	Access	Description
503	Current	r	Return the current field.

14.37.4 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 (497) instance that created the enumerator. It initializes the enumerator position.

14.37.5 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 (503)

14.37.6 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 (502)

14.38 TFloatField

14.38.1 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 (504), MaxValue (504), MinValue (505) and overrides some TField (462) 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 (504), MaxValue (504), MinValue (505)

14.38.2 Method overview

Page	Method	Description
504	CheckRange	Check whether a value is in the allowed range of values for the field.
503	Create	Create a new instance of the `TFloatField`.

14.38.3 Property overview

Page	Properties	Access	Description
504	Currency	rw	Is the field a currency field.
504	MaxValue	rw	Maximum value for the field.
505	MinValue	rw	Minimum value for the field.
505	Precision	rw	Precision (number of digits) of the field in text representations.
504	Value	rw	Value of the field as a double type.

14.38.4 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.

14.38.5 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 (505) and MaxValue (504) properties. If the value lies outside of the allowed range, then `False` is returned.

See also: MaxValue (504), MinValue (505)

14.38.6 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 (470)

See also: TField.AsFloat (470), TField.Value (479)

14.38.7 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 (476) and TField.Text (478) 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 (476), TField.Text (478), TNumericField.DisplayFormat (528), TNumericField.EditFormat (529)

14.38.8 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 (505) should also be set, because it will also be checked.

See also: TFloatField.MinValue (505)

14.38.9 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 (504) should also be set, because it will also be checked.

See also: TFloatField.MaxValue (504), TFloatField.CheckRange (504)

14.38.10 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 (476) or TField.Text (478), it is used in the arguments to FormatFloat (**??**).

See also: TField.Displaytext (476), TField.Text (478), FormatFloat (**??**)

14.39 TFMTBCDField

14.39.1 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 (503)

14.39.2 Method overview

Page	Method	Description
506	CheckRange	Check value if it is in the range defined by MinValue and MaxValue.
506	Create	Create a new instance of the `TFMTBCDField` class.

14.39.3 Property overview

Page	Properties	Access	Description
507	Currency	rw	Does the field contain currency data ?
507	MaxValue	rw	Maximum value for the field.
507	MinValue	rw	Minimum value for the field.
507	Precision	rw	Total number of digits in the BCD data.
508	Size		Number of digits after the decimal point.
506	Value	rw	The value of the field as a BCD value.

14.39.4 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 (351), MaxValue (351), Size (478) (15) and Precision (351) (2) fields to their default values.

See also: MinValue (351), MaxValue (351), Size (478), Precision (351)

14.39.5 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 (351) and MaxValue (351) 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 (351), MaxValue (351)

14.39.6 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 (470), TField.AsCurrency (469)

14.39.7 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 (470), TField.AsCurrency (469), Size (351)

14.39.8 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 (462) or EditFormat (462) are set, these values are used instead to format the value.

See also: TField.DisplayFormat (462), TField.EditFormat (462)

14.39.9 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 (351) or it will not have any effect.

See also: TFMTBCDField.CheckRange (506), MinValue (351)

14.39.10 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 (351) or it will not have any effect.

See also: TFMTBCDField.CheckRange (506), MaxValue (351)

14.39.11 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 (351) property it determines the maximum allowed range of values for the field. This range can be restricted using the MinValue (351) and MaxValue (351) properties.

See also: MinValue (351), MaxValue (351), Precision (351)

14.40 TGraphicField

14.40.1 Description

`TGraphicField` is the class used when a dataset must manage graphical BLOB data. (TField.DataType (475) equals `ftGraphic`). It initializes some of the properties of the TField (462) class. All methods to be able to work with graphical BLOB data have been implemented in the TBlobField (384) 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 (409), TField (462), TBLOBField (384), TMemoField (525), TWideMemoField (558)

14.40.2 Method overview

Page	Method	Description
508	Create	Create a new instance of the `TGraphicField` class.

14.40.3 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 (462) properties to configure the instance for working with graphical BLOB values.

See also: TField (462)

14.41 TGuidField

14.41.1 Description

`TGUIDField` is the class used when a dataset must manage native variant-typed data. (TField.DataType (475) equals `ftGUID`). It initializes some of the properties of the TField (462) 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 (409), TField (462), TGuidField.AsGuid (509)

14.41.2 Method overview

Page	Method	Description
509	Create	Create a new instance of the `TGUIDField` class.

14.41.3 Property overview

Page	Properties	Access	Description
509	AsGuid	rw	Field content as a GUID value.

14.41.4 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 (462) properties to configure the instance for working with GUID values.

See also: TField (462)

14.41.5 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 (472)

14.42 TIndexDef

14.42.1 Description

`TIndexDef` describes one index in a set of indexes of a TDataset (409) instance. The collection of indexes is described by the TIndexDefs (512) class. It just has the necessary properties to describe an index, but does not implement any functionality to maintain an index.

See also: TIndexDefs (512)

14.42.2 Method overview

Page	Method	Description
510	Create	Create a new index definition.

14.42.3 Property overview

Page	Properties	Access	Description
511	CaseInsFields	rw	Fields in field list that are ordered case-insensitively.
511	DescFields	rw	Fields in field list that are ordered descending.
510	Expression	rw	Expression that makes up the index values.
510	Fields	rw	Fields making up the index.
511	Options	rw	Index options.
512	Source	rw	Source of the index.

14.42.4 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 (509) 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 (512) 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 (512), TIndexDef.Options (511), TIndexDef.Fields (510)

14.42.5 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 (511)

See also: TIndexDef.Options (511), TindexDef.Fields (510)

14.42.6 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 (511) property can be used to determine the fields in the list that have a descending sort order. The TIndexDef.CaseInsFields (511) property determines which fields are sorted in a case-insensitive manner.

See also: TIndexDef.DescFields (511), TIndexDef.CaseInsFields (511), TIndexDef.Expression (510)

14.42.7 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 (510) property which are ordered in a case-insensitive manner. `CaseInsFields` may not contain fieldnames that do not appear in `Fields`.

See also: TIndexDef.Fields (510), TIndexDef.Expression (510), TIndexDef.DescFields (511)

14.42.8 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 (510) property which are ordered in a descending manner. `DescFields` may not contain fieldnames that do not appear in `Fields`.

See also: TIndexDef.Fields (510), TIndexDef.Expression (510), TIndexDef.DescFields (511)

14.42.9 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 (363) for more information on the various available options.

See also: TIndexOptions (363)

14.42.10 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.

14.43 TIndexDefs

14.43.1 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 (509), TIndexDefs.Items (514)

14.43.2 Method overview

Page	Method	Description
513	Add	Add a new index definition with given name and options.
513	AddIndexDef	Add a new, empty, index definition.
512	Create	Create a new `TIndexDefs` instance.
513	Find	Find an index by name.
513	FindIndexForFields	Find index definition based on field names.
514	GetIndexForFields	Get index definition based on field names.
514	Update	Called whenever one of the items changes.

14.43.3 Property overview

Page	Properties	Access	Description
514	Items	rw	Indexed access to the index definitions.

14.43.4 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 (509).

See also: TIndexDef (509), TIndexDefs.Destroy (512)

14.43.5 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 (509) 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 (509), TNamedItem.Name (527), TIndexDef.Fields (510), TIndexDef.Options (511), TIndexDefs.AddIndexDef (513)

14.43.6 TIndexDefs.AddIndexDef

Synopsis: Add a new, empty, index definition.

Declaration:
```
function AddIndexDef : TIndexDef
```

Visibility: public

Description: `AddIndexDef` adds a new TIndexDef (509) 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 (513)

14.43.7 TIndexDefs.Find

Synopsis: Find an index by name.

Declaration:
```
function Find(const IndexName: string) : TIndexDef
```

Visibility: public

Description: `Find` overloads the TDefCollection.Find (459) method to search and return a TIndexDef (509) 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 (509), TDefCollection.Find (459), TIndexDefs.FindIndexForFields (513)

14.43.8 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 (510) 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 (509), TIndexDefs.Find (513), TIndexDefs.GetindexForFields (514)

14.43.9 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 (510) 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 (509), TIndexDefs.Find (513), TIndexDefs.FindIndexForFields (513)

14.43.10 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.

14.43.11 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 (509)

14.44 TIntegerField

14.44.1 Description

`TIntegerField` is an alias for TLongintField (516).

See also: TLongintField (516), TField (462)

14.45 TLargeintField

14.45.1 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 (462) 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 (516) and MaxValue (516).

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 (462), MinValue (516), MaxValue (516)

14.45.2 Method overview

Page	Method	Description
515	CheckRange	Check whether a values falls within the allowed range.
515	Create	Create a new instance of the `TLargeintField` class.

14.45.3 Property overview

Page	Properties	Access	Description
516	MaxValue	rw	Maximum value for the field.
516	MinValue	rw	Minimum value for the field.
515	Value	rw	Field contents as a 64-bit integer value.

14.45.4 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 (462) and MinValue (516) and MaxValue (516).

See also: TField (462), MinValue (516), MaxValue (516)

14.45.5 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 (516) and MaxValue (516) properties. If the value lies outside of the allowed range, then `False` is returned.

See also: MaxValue (516), MinValue (516)

14.45.6 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 (471).

See also: TField.Value (479), TField.AsLargeInt (471)

14.45.7 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 (516) should also be set, because it will also be checked.

See also: TLargeIntField.MinValue (516)

14.45.8 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 (516) should also be set, because it will also be checked.

See also: TLargeIntField.MaxValue (516)

14.46 TLongintField

14.46.1 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 (462) 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 (518) and MaxValue (518).

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 (462), MaxValue (518), MinValue (518)

14.46.2 Method overview

Page	Method	Description
517	CheckRange	Check whether a valid is in the allowed range of values for the field.
517	Create	Create a new instance of `TLongintField`.

14.46.3 Property overview

Page	Properties	Access	Description
518	MaxValue	rw	Maximum value for the field.
518	MinValue	rw	Minimum value for the field.
517	Value	rw	Value of the field as longint.

14.46.4 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 (518) and MaxValue (518) properties.

See also: TField (462), MaxValue (518), MinValue (518)

14.46.5 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 (518) and MaxValue (518) properties. If the value lies outside of the allowed range, then `False` is returned.

See also: MaxValue (518), MinValue (518)

14.46.6 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 (471) property.

See also: TField.Value (479)

14.46.7 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 (518)

14.46.8 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 (518)

14.47 TLongWordField

14.47.1 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 (462) 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 (520) and MaxValue (519).

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 (520), MaxValue (519)

14.47.2 Method overview

Page	Method	Description
519	CheckRange	Checkif a value is in the allowed range.
519	Create	Create new instance of `TLongWordField`.

14.47.3 Property overview

Page	Properties	Access	Description
519	MaxValue	rw	Maximum field value.
520	MinValue	rw	Minimum field value.
519	Value	rw	Value as longword.

14.47.4 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 (520)MaxValue (519) and TField.DataType (475) properties.

See also: MinValue (518), MaxValue (518), TField.DataType (475)

14.47.5 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 (520) or MaxValue (519)

See also: MinValue (520), MaxValue (519)

14.47.6 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 (479)

14.47.7 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 (520), CheckRange (519)

14.47.8 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 (519), CheckRange (519)

14.48 TLookupList

14.48.1 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 (484)

14.48.2 Method overview

Page	Method	Description
521	Add	Add a key, value pair to the list.
521	Clear	Remove all key, value pairs from the list.
520	Create	Create a new instance of `TLookupList`.
520	Destroy	Free a `TLookupList` instance from memory.
521	FirstKeyByValue	Find the first key that matches a value.
521	ValueOfKey	Look up value based on a key.
522	ValuesToStrings	Convert values to stringlist.

14.48.3 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 (520)

14.48.4 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 (520)

14.48.5 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 (521)

14.48.6 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 (521)

14.48.7 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 (521)

14.48.8 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 (521), TLookupList.Add (521)

14.48.9 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 (521)

14.49 TMasterDataLink

14.49.1 Description

`TMasterDataLink` is a `TDatalink` descendent which handles master-detail relations. It can be used in TDataset (409) 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 (408) property.

The class also provides functionality for keeping a list of fields that make up the master-detail functionality, in the TMasterDatalink.FieldNames (523) and TMasterDataLink.Fields (523) properties.

This class should never be used in application code.

See also: TDataset (409), TDatalink.DataSource (408), TDatalink.DataSet (408), TMasterDatalink.FieldNames (523), TMasterDataLink.Fields (523)

14.49.2 Method overview

Page	Method	Description
522	Create	Create a new instance of `TMasterDataLink`.
523	Destroy	Free the datalink instance from memory.

14.49.3 Property overview

Page	Properties	Access	Description
523	FieldNames	rw	List of fieldnames that make up the master-detail relationship.
523	Fields	r	List of fields as specified in `FieldNames`.
523	OnMasterChange	rw	Called whenever the master dataset data changes.
524	OnMasterDisable	rw	Called whenever the master dataset is disabled.

14.49.4 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 (460) property. The master dataset must be set through the DataSource (408) property, and is usually set by the application programmer.

See also: TDetailDataLink.DetailDataset (460), TDatalink.Datasource (408)

14.49.5 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 (522)

14.49.6 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 (408)) on which the master-detail relationship is based. Setting this property will fill the TMasterDataLink.Fields (523) property with the field instances of the master dataset.

See also: TMasterDataLink.Fields (523), TDatalink.Dataset (408), TDataset.GetFieldList (424)

14.49.7 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 (462) instances from the master dataset (TDatalink.Dataset (408)) when the FieldNames (523) property is set, and when the master dataset opens.

See also: TField (462), TMasterDatalink.FieldNames (523)

14.49.8 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 (523) list is empty, TMasterDataLink.OnMasterDisable (524) is called instead.

See also: TMasterDataLink.OnMasterDisable (524)

14.49.9 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 (523)

14.50 TMasterParamsDataLink

14.50.1 Description

`TMasterParamsDataLink` is a TDataLink (404) 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 (404), TDataset (409), TParams (544), TParam (530)

14.50.2 Method overview

Page	Method	Description
525	CopyParamsFromMaster	Copy parameter values from master dataset.
524	Create	Initialize a new `TMasterParamsDataLink` instance.
525	RefreshParamNames	Refresh the list of parameter names.

14.50.3 Property overview

Page	Properties	Access	Description
525	Params	rw	Parameters of detail dataset.

14.50.4 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 (544) in the published properties of `ADataset` and assigns it to the Params (525) property.

See also: TDataset (409), TParams (544), TMasterParamsDataLink.Params (525)

14.50.5 TMasterParamsDataLink.RefreshParamNames

Synopsis: Refresh the list of parameter names.

Declaration: `procedure RefreshParamNames;  Virtual`

Visibility: public

Description: `RefreshParamNames` scans the Params (525) property and sets the FieldNames (523) property to the list of parameter names.

See also: TMasterParamsDataLink.Params (525), TMasterDataLink.FieldNames (523)

14.50.6 TMasterParamsDataLink.CopyParamsFromMaster

Synopsis: Copy parameter values from master dataset.

Declaration: `procedure CopyParamsFromMaster(CopyBound: Boolean);  Virtual`

Visibility: public

Description: `CopyParamsFromMaster` calls TParams.CopyParamValuesFromDataset (549), 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 (541) 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 (549), TParam.Bound (541)

14.50.7 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 (544), TMasterParamsDataLink.Create (524)

14.51 TMemoField

14.51.1 Description

`TMemoField` is the class used when a dataset must manage memo (Text BLOB) data. (TField.DataType (475) equals `ftMemo`). It initializes some of the properties of the TField (462) class. All methods to be able to work with memo fields have been implemented in the TBlobField (384) 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 (409), TField (462), TBLOBField (384), TWideMemoField (558), TGraphicField (508)

14.51.2 Method overview

Page	Method	Description
526	Create	Create a new instance of the `TMemoField` class.

14.51.3 Property overview

Page	Properties	Access	Description
526	CodePage	r	Codepage of the memo field string data.
526	Transliterate		Should the contents of the field be transliterated.

14.51.4 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 (462) properties to configure the instance for working with memo values.

See also: TField (462)

14.51.5 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 (472), TFieldDef.CodePage (492)

14.51.6 TMemoField.Transliterate

Synopsis: Should the contents of the field be transliterated.

Declaration: `Property Transliterate :`

Visibility: published

Access:

Description: `Transliterate` is redefined from TBlobField.Transliterate (388) with a default value of `true`.

See also: TBlobField.Transliterate (388), TStringField.Transliterate (555), TDataset.Translate (430)

14.52 TNamedItem

14.52.1 Description

NamedItem is a TCollectionItem (**??**) descendent which introduces a Name (527) property. It automatically returns the value of the Name property as the value of the DisplayName (527) property.

See also: DisplayName (527), Name (527)

14.52.2 Property overview

Page	Properties	Access	Description
527	DisplayName	rw	Display name.
527	Name	rw	Name of the item.

14.52.3 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 (527) property.

See also: Name (527)

14.52.4 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 (527) property. If the TNamedItem item is owned by a TDefCollection (458) collection, then the name must be unique, i.e. each Name value may appear only once in the collection.

See also: DisplayName (527), TDefCollection (458)

14.53 TNumericField

14.53.1 Description

TNumericField is an abstract class which overrides some of the methods of TField (462) 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 (528) and TNumericField.EditFormat (529).

Since TNumericField is an abstract class, it must never be instantiated directly. Instead one of the descendent classes should be created.

See also: TField (462), TNumericField.DisplayFormat (528), TNumericField.EditFormat (529), TField.Alignment (480), TIntegerField (514), TLargeIntField (514), TFloatField (503), TBCDField (380)

14.53.2 Method overview

Page	Method	Description
528	Create	Create a new instance of `TNumericField`.

14.53.3 Property overview

Page	Properties	Access	Description
528	Alignment		Alignment of the field.
528	DisplayFormat	rw	Format string for display of numerical data.
529	EditFormat	rw	Format string for editing of numerical data.

14.53.4 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 (480) property with

See also: TField.Alignment (480)

14.53.5 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 (480)

14.53.6 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 (476) 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 (476), TNumericField.EditFormat (529)

14.53.7 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 (478) 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 (478), TNumericField.DisplayFormat (528)

14.54 TObjectField

14.54.1 Property overview

Page	Properties	Access	Description
529	FieldCount	r	
529	Fields	r	
529	FieldValues	rw	
530	ObjectType	rw	
530	UnNamed	r	

14.54.2 TObjectField.FieldCount

Declaration: `Property FieldCount : Integer`

Visibility: public

Access: Read

14.54.3 TObjectField.Fields

Declaration: `Property Fields : TFields`

Visibility: public

Access: Read

14.54.4 TObjectField.FieldValues

Declaration: `Property FieldValues[AIndex: Integer]: Variant; default`

Visibility: public

Access: Read,Write

14.54.5 TObjectField.UnNamed

Declaration: `Property UnNamed : Boolean`

Visibility: public

Access: Read

14.54.6 TObjectField.ObjectType

Declaration: `Property ObjectType : string`

Visibility: published

Access: Read,Write

14.55 TParam

14.55.1 Description

`TParam` is one item in a TParams (544) collection. It describes the name (TParam.Name (542)), type (ParamType (543)) and value (TParam.Value (544)) 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 (409) descendent that owns the parameters should have created all necessary `TParam` instances.

See also: TParams (544)

14.55.2 Method overview

Page	Method	Description
532	Assign	Assign one parameter instance to another.
532	AssignField	Copy value from field instance.
532	AssignFieldValue	Assign field value to the parameter.
533	AssignFromField	Copy field type and value.
532	AssignToField	Assign parameter value to field.
533	Clear	Clear the parameter value.
531	Create	Create a new parameter value.
533	GetData	Get the parameter value from a memory buffer.
533	GetDataSize	Return the size of the data.
534	LoadFromFile	Load a parameter value from file.
534	LoadFromStream	Load a parameter value from stream.
534	SetBlobData	Set BLOB data.
534	SetData	Set the parameter value from a buffer.

14.55.3 Property overview

Page	Properties	Access	Description
539	AsAnsiString	rw	Parameter contents as an ANSI string.
535	AsBCD	rw	Get or set parameter value as BCD value.
535	AsBlob	rw	Return parameter value as a blob.
535	AsBoolean	rw	Get/Set parameter value as a boolean value.
536	AsByte	rw	Get/Set parameter value as a 8-bit unsigned integer value.
536	AsBytes	rw	Get or set parameter value as `TBytes`.
536	AsCurrency	rw	Get/Set parameter value as a currency value.
536	AsDate	rw	Get/Set parameter value as a date (TDateTime) value.
537	AsDateTime	rw	Get/Set parameter value as a date/time (TDateTime) value.
537	AsFloat	rw	Get/Set parameter value as a floating-point value.
540	AsFMTBCD	rw	Parameter value as a BCD value.
537	AsInteger	rw	Get/Set parameter value as an integer (32-bit) value.
537	AsLargeInt	rw	Get/Set parameter value as a 64-bit integer value.
538	AsLongWord	rw	Get/Set parameter value as a 32-bit unsigned integer value.
538	AsMemo	rw	Get/Set parameter value as a memo (string) value.
538	AsShortInt	rw	
538	AsSingle	rw	
538	AsSmallInt	rw	Get/Set parameter value as a smallint value.
539	AsString	rw	Get/Set parameter value as a string value.
540	AsTime	rw	Get/Set parameter value as a time (TDateTime) value.
540	AsUnicodeString	rw	Parameter contents as a Unicode string.
539	AsUTF8String	rw	Parameter contents as an UTF8 string.
542	AsWideString	rw	Get/Set the value as a widestring.
540	AsWord	rw	Get/Set parameter value as a word value.
541	Bound	rw	Is the parameter value bound (set to fixed value).
541	Dataset	r	Dataset to which this parameter belongs.
542	DataType	rw	Data type of the parameter.
541	IsNull	r	Is the parameter empty.
542	Name	rw	Name of the parameter.
541	NativeStr	rw	No description available.
543	NumericScale	rw	Numeric scale.
543	ParamType	rw	Type of parameter.
543	Precision	rw	Precision of the BCD value.
544	Size	rw	Size of the parameter.
542	Text	rw	Read or write the value of the parameter as a string.
544	Value	rws	Value as a variant.

14.55.4 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 (544)

14.55.5 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 (462), then it is passed to TParam.AssignField (532). If `Source` is of type `TStrings`, then it is assigned to TParams.AsMemo (544).

Errors: If `Source` is not of type `TParam`, `TField` or `TStrings`, an exception will be raised.

See also: TField (462), TParam.Name (542), TParam.Bound (541), TParam.NumericScale (543), TParam.ParamType (543), TParam.value (544), TParam.Size (544), TParam.AssignField (532), Tparam.AsMemo (538)

14.55.6 TParam.AssignField

Synopsis: Copy value from field instance.

Declaration: `procedure AssignField(Field: TField)`

Visibility: public

Description: `AssignField` copies the `Field`, FieldName (482) and Value (479) 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 (532), TParam.AssignToField (532), TParam.AssignFieldValue (532)

14.55.7 TParam.AssignToField

Synopsis: Assign parameter value to field.

Declaration: `procedure AssignToField(Field: TField)`

Visibility: public

Description: `AssignToField` copies the parameter value (544) to the `field` instance. If `Field` is `Nil`, nothing happens.

Errors: An EDatabaseError (371) exception is raised if the field has an unsupported field type (for types ftCursor, ftArray, ftDataset,ftReference).

See also: TParam.Assign (532), TParam.AssignField (532), TParam.AssignFromField (533)

14.55.8 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 (541) bound parameter to `True`. This method is called from TParam.AssignField (532).

See also: TField (462), TParam.AssignField (532), TParam.Bound (541)

14.55.9 TParam.AssignFromField

Synopsis: Copy field type and value.

Declaration: `procedure AssignFromField(Field: TField)`

Visibility: public

Description: `AssignFromField` copies the field value (479) and data type (TField.DataType (475)) to the parameter instance. If `Field` is `Nil`, nothing happens. This is the reverse operation of TParam.AssignToField (532).

Errors: An EDatabaseError (371) exception is raised if the field has an unsupported field type (for types ftCursor, ftArray, ftDataset,ftReference).

See also: TParam.Assign (532), TParam.AssignField (532), TParam.AssignToField (532)

14.55.10 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 (544), TParam.Name (542), TParam.ParamType (543), TParam.DataType (542)

14.55.11 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 (462), 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 (371) exception.

See also: TParam.SetData (534), TField (462)

14.55.12 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 (371) exception is raised when this function is called.

See also: TParam.GetData (533), TParam.SetData (534)

14.55.13 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 (534).

Errors: If the specified `FileName` is not a valid file, or the file is not readable, an exception will occur.

See also: TParam.LoadFromStream (534), TBlobType (354), TParam.SaveToFile (530)

14.55.14 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 (534), TParam.SaveToStream (530)

14.55.15 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 (534)

14.55.16 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 (533): it reads the parameter value from the memory area pointed to by `Buffer`. The size of the data read is determined by TParam.GetDataSize (533) and the type of data by TParam.DataType (542) : it is the same storage mechanism used by TField (462), and so can be used to copy the value from a TDataset (409) record buffer.

Errors: Not all field types are supported. If an unsupported field type is encountered, an EDatabaseError (371) exception is raised.

See also: TDataset (409), TParam.GetData (533), TParam.DataType (542), TParam.GetDataSize (533)

14.55.17 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 (537)

14.55.18 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 (539)

14.55.19 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 (542), TParam.Value (544)

14.55.20 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 (537), TParam.asSmallint (538), TParam.AsWord (540), TParam.DataType (542), TParam.Value (544)

14.55.21 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 (539)

14.55.22 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 (537), TParam.DataType (542), TParam.Value (544)

14.55.23 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 (537), TParam.AsTime (540), TParam.DataType (542), TParam.Value (544)

14.55.24 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 (536), TParam.asTime (540), TParam.DataType (542), TParam.Value (544)

14.55.25 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 (536), TParam.DataType (542), TParam.Value (544)

14.55.26 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 (537), TParam.AsSmallInt (538), TParam.AsWord (540), TParam.DataType (542), TParam.Value (544)

14.55.27 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 (537), TParam.asSmallint (538), TParam.AsWord (540), TParam.DataType (542), TParam.Value (544)

14.55.28 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 (537), TParam.asSmallint (538), TParam.AsWord (540), TParam.DataType (542), TParam.Value (544)

14.55.29 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 (539), TParam.LoadFromStream (534), TParam.SaveToStream (530), TParam.DataType (542), TParam.Value (544)

14.55.30 TParam.AsShortInt

Declaration: `Property AsShortInt : LongInt`

Visibility: public

Access: Read,Write

14.55.31 TParam.AsSingle

Declaration: `Property AsSingle : Single`

Visibility: public

Access: Read,Write

14.55.32 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 (537), TParam.AsLargeInt (537), TParam.AsWord (540), TParam.DataType (542), TParam.Value (544)

14.55.33 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 (542), TParam.Value (544)

14.55.34 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 (539), TParam.AsUnicodeString (540), TParam.AsUTF8String (539)

14.55.35 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 (539), TParam.AsUnicodeString (540), TParam.AsAnsiString (539)

14.55.36 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 (539), TParam.AsUTF8String (539), TParam.AsAnsiString (539)

14.55.37 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 (536), TParam.AsDateTime (537), TParam.DataType (542), TParam.Value (544)

14.55.38 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 (537), TParam.AsLargeInt (537), TParam.AsSmallint (538), TParam.DataType (542), TParam.Value (544)

14.55.39 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 (351), AsCurrency (351)

14.55.40 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 (542), TParam.Value (544)

14.55.41 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 (544) 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 (409), TParams (544)

14.55.42 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 (533), TParam.Value (544)

14.55.43 TParam.NativeStr

Synopsis: No description available.

Declaration: `Property NativeStr : string`

Visibility: public

Access: Read,Write

Description: No description available

14.55.44 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 (539), but, when written, does not set the data type: instead, it attempts to convert the value to the type specified in TParam.Datatype (542).

See also: TParam.AsString (539), TParam.DataType (542)

14.55.45 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 (542) to `ftWideString`.

See also: TParam.AsString (539), TParam.Value (544), TParam.DataType (542)

14.55.46 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 (541), TParam.Value (544), TParam.AssignField (532)

14.55.47 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 (542), TParam.Value (544), TParams.ParamByName (547)

14.55.48 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 (543), TParam.Size (544)

14.55.49 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 (542), TParam.DataSize (530), TParam.Name (542)

14.55.50 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 (543), TParam.Size (544)

14.55.51 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 (475) in the TParam.AssignFieldValue (532) method. The actual size can be retrieved through the TParam.Datasize (530) property.

See also: TParam.Datasize (530), TField.DataSize (475), TParam.AssignFieldValue (532)

14.55.52 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 (542)

14.56 TParams

14.56.1 Description

`TParams` is a collection of TParam (530) 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 (530) 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 (547).

TDataset (409) 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 (524) 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 (549) method exists.

See also: TDataset (409), TMasterParamsDataLink (524), TParam (530), TParams.ParseSQL (547), TParams.CopyParamValuesFromDa (549)

14.56.2 Method overview

Page	Method	Description
545	AddParam	Add a parameter to the collection.
546	AssignValues	Copy values from another collection.
549	CopyParamValuesFromDataset	Copy parameter values from the fields in a dataset.
545	Create	Create a new instance of `TParams`.
546	CreateParam	Create and add a new parameter to the collection.
546	FindParam	Find a parameter with given name.
547	GetEnumerator	Return an enumerator for the parameters.
546	GetParamList	Fetch a list of `TParam` instances.
547	IsEqual	Is the list of parameters equal.
547	ParamByName	Return a parameter by name.
547	ParseSQL	Parse SQL statement, replacing parameter names with SQL parameter placeholders.
548	RemoveParam	Remove a parameter from the collection.

14.56.3 Property overview

Page	Properties	Access	Description
549	Dataset	r	Dataset that owns the `TParams` instance.
549	Items	rw	Indexed access to `TParams` instances in the collection.
549	ParamValues	rw	Named access to the parameter values.

14.56.4 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 (530) 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 (530)

14.56.5 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 (530), #rtl.classes.tcollection.add (**??**)

14.56.6 TParams.AssignValues

Synopsis: Copy values from another collection.

Declaration:
```
procedure AssignValues(Value: TParams)
```

Visibility: public

Description: `AssignValues` examines all TParam (530) 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 (532)). If it is not found, nothing is done.

See also: TParam (530), TParam.Assign (532)

14.56.7 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 (530) 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 (530), TParam.Name (542), TParam.Datatype (542), TParam.Paramtype (543)

14.56.8 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 (530) 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 (542), TParams.ParamByName (547), TParams.GetParamList (546)

14.56.9 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 (530), TParam.Name (542), TParams.ParamByName (547)

14.56.10 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 (530)

14.56.11 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 (550))

See also: TParamsEnumerator (550)

14.56.12 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 (530) 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 (542), TParams.FindParam (546), TParams.GetParamList (546)

14.56.13 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 (530), TParam.Name (542), TParamStyle (364)

14.56.14 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 (530)

14.56.15 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 (532). By default, this operation is only performed on `TParam` instances with their Bound (541) 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 (530), TParam.Bound (541), TParam.AssignField (532), TDataset (409), TDataset.FieldByName (421)

14.56.16 TParams.Dataset

Synopsis: Dataset that owns the `TParams` instance.

Declaration:
```
Property Dataset : TDataSet
```

Visibility: public

Access: Read

Description: `Dataset` is the TDataset (409) instance that was specified when the `TParams` instance was created.

See also: TParams.Create (545), TDataset (409)

14.56.17 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 (544)

14.56.18 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 (544)) by name. It is equivalent to reading and writing

```
ParamByName(ParamName).Value
```

See also: TParam.Value (544), TParams.ParamByName (547)

14.57 TParamsEnumerator

14.57.1 Description

`TParamsEnumerator` is a helper class to implement enumeration (`for..in`) of parameters. It implements the `IEnumerator` interface.

See also: TParams.GetEnumerator (547)

14.57.2 Method overview

Page	Method	Description
550	Create	Create a new `TParamsEnumerator` instance.
550	MoveNext	Go to next TParam.

14.57.3 Property overview

Page	Properties	Access	Description
551	Current	r	Current `TParam` instance.

14.57.4 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 (547)

14.57.5 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 (551) will return the new current `TParam`

See also: TParamsEnumerator.Current (551)

14.57.6 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 (550) returned true.

See also: TParamsEnumerator.MoveNext (550)

14.58 TShortintField

14.58.1 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 (462) 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 (518) and MaxValue (518).

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 (518), MaxValue (518)

14.58.2 Method overview

Page	Method	Description
551	Create	Create new instance of `TShortintField`.

14.58.3 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 (518)MaxValue (518) and TField.DataType (475) properties.

See also: MinValue (518), MaxValue (518), TField.DataType (475)

14.59 TSingleField

14.59.1 Method overview

Page	Method	Description
552	CheckRange	
552	Create	

14.59.2 Property overview

Page	Properties	Access	Description
552	Currency	rw	
552	MaxValue	rw	
552	MinValue	rw	
553	Precision	rw	
552	Value	rw	

14.59.3 TSingleField.Create

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

Visibility: public

14.59.4 TSingleField.CheckRange

Declaration: `function CheckRange(AValue: Single) : Boolean`

Visibility: public

14.59.5 TSingleField.Value

Declaration: `Property Value : Single`

Visibility: public

Access: Read,Write

14.59.6 TSingleField.Currency

Declaration: `Property Currency : Boolean`

Visibility: published

Access: Read,Write

14.59.7 TSingleField.MaxValue

Declaration: `Property MaxValue : Single`

Visibility: published

Access: Read,Write

14.59.8 TSingleField.MinValue

Declaration: `Property MinValue : Single`

Visibility: published

Access: Read,Write

14.59.9 TSingleField.Precision

Declaration: `Property Precision : LongInt`

Visibility: published

Access: Read,Write

14.60 TSmallintField

14.60.1 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 (462), TNumericField (527), TLongintField (516), TWordField (560)

14.60.2 Method overview

Page	Method	Description
553	Create	Create a new instance of the `TSmallintField` class.

14.60.3 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 (553) class. It calls the inherited constructor and then simply sets some of the TField (462) properties to work with 16-bit signed integer data.

See also: TField (462)

14.61 TStringField

14.61.1 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 (462) 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 (462), TWideStringField (559), TDataset (409)

14.61.2 Method overview

Page	Method	Description
554	Create	Create a new instance of the TStringField class.
554	SetFieldType	Set the field type.

14.61.3 Property overview

Page	Properties	Access	Description
554	CodePage	r	Codepage of the field string data.
555	EditMask		Specify an edit mask for an edit control.
555	FixedChar	rw	Is the string declared with a fixed length ?
556	Size		Maximum size of the string.
555	Transliterate	rw	Should the field value be transliterated when reading or writing.
555	Value	rw	Value of the field as a string.

14.61.4 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 (462) properties after having called the inherited constructor.

14.61.5 TStringField.SetFieldType

Synopsis: Set the field type.

Declaration: `procedure SetFieldType(AValue: TFieldType);  Override`

Visibility: public

Description: `SetFieldType` is overridden in TStringField (553) to check the data type more accurately (`ftString` and `ftFixedChar`). No extra functionality is added.

See also: TField.DataType (475)

14.61.6 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 (472), TField.AsUnicodeString (473), TField.AsAnsi8String (462), TFieldDef.CodePage (492)

14.61.7 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.

14.61.8 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 (430).

See also: TDataset.Translate (430)

14.61.9 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 (472) when read, or sets the `AsString` property when written to.

See also: TField.AsString (472), TField.Value (479)

14.61.10 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 (476)

14.61.11 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 (478)

14.62 TTimeField

14.62.1 Description

`TimeField` is the class used when a dataset must manage data of type time. (TField.DataType (475) equals `ftTime`). It initializes some of the properties of the TField (462) 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 (409), TField (462), TDateTimeField (453), TDateField (453)

14.62.2 Method overview

Page	Method	Description
556	Create	Create a new instance of a `TTimeField` class.

14.62.3 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 (462) properties to configure the instance for working with time values.

See also: TField (462)

14.63 TVarBytesField

14.63.1 Description

`TVarBytesField` is the class used when a dataset must manage data of variable-size binary type. (TField.DataType (475) equals `ftVarBytes`). It initializes some of the properties of the TField (462) 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 (409), TField (462), TBytesField (391)

14.63.2 Method overview

Page	Method	Description
557	Create	Create a new instance of a `TVarBytesField` class.

14.63.3 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 (462) properties to configure the instance for working with variable-size binary data values.

See also: TField (462)

14.64 TVariantField

14.64.1 Description

`TVariantField` is the class used when a dataset must manage native variant-typed data. (TField.DataType (475) equals `ftVariant`). It initializes some of the properties of the TField (462) 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 (409), TField (462)

14.64.2 Method overview

Page	Method	Description
558	Create	Create a new instance of the `TVariantField` class.

14.64.3 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 (462) properties to configure the instance for working with variant values.

See also: TField (462)

14.65 TWideMemoField

14.65.1 Description

`TWideMemoField` is the class used when a dataset must manage memo (Text BLOB) data. (TField.DataType (475) equals `ftWideMemo`). It initializes some of the properties of the TField (462) class. All methods to be able to work with widestring memo fields have been implemented in the TBlobField (384) 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 (409), TField (462), TBLOBField (384), TMemoField (525), TGraphicField (508)

14.65.2 Method overview

Page	Method	Description
558	Create	Create a new instance of the `TWideMemoField` class.

14.65.3 Property overview

Page	Properties	Access	Description
559	Value	rw	Value of the field's contents as a widestring.

14.65.4 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 (462) properties to configure the instance for working with widestring memo values.

See also: TField (462)

14.65.5 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 (473) property.

See also: TField.Value (479), Tfield.AsWideString (473)

14.66 TWideStringField

14.66.1 Description

`TWideStringField` is the string field class instantiated for fields of data type `ftWideString`. This class overrides some of the standard TField (462) 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 (462), TStringField (553), TDataset (409)

14.66.2 Method overview

Page	Method	Description
559	Create	Create a new instance of the `TWideStringField` class.
559	SetFieldType	Set the field type.

14.66.3 Property overview

Page	Properties	Access	Description
560	Value	rw	Value of the field as a widestring.

14.66.4 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 (462) properties after having called the inherited constructor.

14.66.5 TWideStringField.SetFieldType

Synopsis: Set the field type.

Declaration: `procedure SetFieldType(AValue: TFieldType); Override`

Visibility: public

Description: `SetFieldType` is overridden in TWideStringField (559) to check the data type more accurately (`ftWideString` and `ftFixedWideChar`). No extra functionality is added.

See also: TField.DataType (475)

14.66.6 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 (473) property.

See also: TField.AsWideString (473), TField.Value (479)

14.67 TWordField

14.67.1 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 (462), TNumericField (527), TLongintField (516), TSmallIntField (553)

14.67.2 Method overview

Page	Method	Description
560	Create	Create a new instance of the `TWordField` class.

14.67.3 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 (560) class. It calls the inherited constructor and then simply sets some of the TField (462) properties to work with 16-bit unsigned integer data.

See also: TField (462)

Chapter 15

Reference for unit 'dbugintf'

15.1 Used units

Table 15.1: Used units by unit 'dbugintf'

Name	Page
dbugmsg	568
System	??

15.2 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 (561) 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 (561) unit should also be functional.

15.3 Writing a debug server.

Writing a debug server is relatively easy. It should instantiate a `TSimpleIPCServer` class from the SimpleIPC (561) 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 (561) 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 (561) unit should also be functional.

15.4 Constants, types and variables

15.4.1 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.

15.4.2 Types

```
TDebugLevel = (dlInformation,dlWarning,dlError)
```

Table 15.2: 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.

15.4.3 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.

15.5 Procedures and functions

15.5.1 FreeDebugClient

Synopsis:

Declaration:
```
procedure FreeDebugClient
```

Visibility: default

Description:

15.5.2 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 (567), SendDebug (564)

15.5.3 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 (564) call will initialize the debug client when it is first called.

Errors: None.

See also: SendDebug (564), StartDebugServer (567)

15.5.4 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 (564) which sends the name and value of a boolean value as an informational message.

Errors: None.

See also: SendDebug (564), SendDateTime (564), SendInteger (565), SendPointer (566)

15.5.5 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 (564) 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 (564), SendBoolean (564), SendInteger (565), SendPointer (566)

15.5.6 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 (563).

See also: SendDebugEx (564), SendDebugFmt (565), SendDebugFmtEx (565)

15.5.7 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 (564) uses informational messages.

Other than that the function of `SendDebugEx` is equal to that of `SendDebug`

Errors: None.

See also: SendDebug (564), SendDebugFmt (565), SendDebugFmtEx (565)

15.5.8 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 (564). It exists mainly to avoid the `Format` call in calling code.

Errors: None.

See also: SendDebug (564), SendDebugEx (564), SendDebugFmtEx (565), #rtl.sysutils.format (**??**)

15.5.9 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 (564) with Debug level `MType`. It exists mainly to avoid the `Format` call in calling code.

Errors: None.

See also: SendDebug (564), SendDebugEx (564), SendDebugFmt (565), #rtl.sysutils.format (**??**)

15.5.10 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 (564) 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 (564), SendBoolean (564), SendDateTime (564), SendPointer (566)

15.5.11 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 (566), 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 (564), SendMethodExit (566), SendSeparator (567)

15.5.12 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 (566), 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 (564), SendMethodEnter (566), SendSeparator (567)

15.5.13 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 (564) 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 (564), SendBoolean (564), SendDateTime (564), SendInteger (565)

15.5.14 SendSeparator

Synopsis: Send a separator message.

Declaration: `procedure SendSeparator`

Visibility: default

Description: `SendSeparator` is a simple wrapper around SendDebug (564) 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 (564), SendMethodEnter (566), SendMethodExit (566)

15.5.15 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 (564) 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 (563), SendDebug (564)

15.5.16 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 (564) call will attempt to start the server by itself if it is not yet running.

Errors: On error, `False` is returned.

See also: SendDebug (564), InitDebugClient (563)

Chapter 16

Reference for unit 'dbugmsg'

16.1 Used units

Table 16.1: Used units by unit 'dbugmsg'

Name	Page
Classes	??
System	??

16.2 Overview

dbugmsg is an auxiliary unit used in the dbugintf (561) unit. It defines the message protocol used between the debug unit and the debug server.

16.3 Constants, types and variables

16.3.1 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.

16.3.2 Types

16.4 Procedures and functions

16.4.1 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 (570) 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.

16.4.2 ReadDebugMessageFromStream

Synopsis: Read a message from stream.

Declaration:
```
procedure ReadDebugMessageFromStream(AStream: TStream;
                                     var Msg: TDebugMessage)
```

Visibility: default

Description: `ReadDebugMessageFromStream` reads a TDebugMessage (570) 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 (570), WriteDebugMessageToStream (569)

16.4.3 WriteDebugMessageToStream

Synopsis: Write a message to stream.

Declaration:
```
procedure WriteDebugMessageToStream(AStream: TStream;
                                    const Msg: TDebugMessage)
```

Visibility: default

Description: `WriteDebugMessageFromStream` writes a TDebugMessage (570) 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 (570), ReadDebugMessageFromStream (569)

16.5 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 (569) and ReadDebugMessageFromStream (569) can be used to read or write the message from/to a stream.

Chapter 17

Reference for unit 'eventlog'

17.1 Used units

Table 17.1: Used units by unit 'eventlog'

Name	Page
Classes	??
System	??
sysutils	??

17.2 Overview

The EventLog unit implements the TEventLog (573) 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.

17.3 Constants, types and variables

17.3.1 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.

17.3.2 Types

```
TLogCategoryEvent = procedure(Sender: TObject; var Code: Word) of
  object
```

`TLogCategoryEvent` is the event type for the TEventLog.OnGetCustomCategory (580) 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 (581) and OnGetCustomEventID (581) 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 (581). If you write your own log message handling method, then it must use this signature.

```
TLogType = (ltSystem,ltFile,ltStdOut,ltStdErr)
```

Table 17.2: 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 (578) 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.

17.4 ELogError

17.4.1 Description

`ELogError` is the exception used in the TEventLog (573) component to indicate errors.

See also: TEventLog (573)

17.5 TEventLog

17.5.1 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 (576) call, or the specialized Warning (576), Error (577), Info (577) or Debug (577) calls, which have the event type predefined.

See also: Log (576), Warning (576), Error (577), Info (577), Debug (577)

17.5.2 Method overview

Page	Method	Description
577	Debug	Log a debug message.
574	Destroy	Clean up `TEventLog` instance.
577	Error	Log an error message to.
574	EventTypeToString	Create a string representation of an event type.
577	Info	Log an informational message.
576	Log	Log a message to the system log.
576	Pause	Pause the sending of log messages.
574	RegisterMessageFile	Register message file.
576	Resume	Resume sending of log messages if sending was paused.
575	UnRegisterMessageFile	Unregister the message file (needed on windows only).
576	Warning	Log a warning message.

17.5.3 Property overview

Page	Properties	Access	Description
578	Active	rw	Activate the log mechanism.
577	AppendContent	rw	Control whether output is appended to an existing file.
580	CustomLogType	rw	Custom log type ID.
579	DefaultEventType	rw	Default event type for the Log (576) call.
580	EventIDOffset	rw	Offset for event ID messages identifiers.
579	FileName	rw	File name for log file.
578	Identification	rw	Identification string for messages.
578	LogType	rw	Log type.
580	OnGetCustomCategory	rw	Event to retrieve custom message category.
581	OnGetCustomEvent	rw	Event to retrieve custom event Code.
581	OnGetCustomEventID	rw	Event to retrieve custom event ID.
581	OnLogMessage	rw	Implement custom handling of log messages.
581	Paused	rw	Is the message sending paused ?
579	RaiseExceptionOnError	rw	Determines whether logging errors are reported or ignored.
579	TimeStampFormat	rw	Format for the timestamp string.

17.5.4 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 (578) property to False.

See also: Active (578)

17.5.5 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 (**??**)

17.5.6 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

1Should contain the description of the `etInfo` event type.

2Should contain the description of the `etWarning` event type.

4Should contain the description of the `etError` event type.

4Should 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 (580) (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.

17.5.7 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 (574). 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 (574)

17.5.8 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 (576).

See also: TEventLog.Resume (576), TEventLog.Paused (581)

17.5.9 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 (571).

See also: TEventLog.Pause (576), TEventLog.Paused (581)

17.5.10 TEventLog.Log

Synopsis: Log a message to the system log.

Declaration:
```
procedure Log(EventType: TEventType; const Msg: string)
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 (579).

If `EventType` is `etCustom`, then the OnGetCustomEvent (581), OnGetCustomEventID (581) and OnGetCustomCategory (580).

The other logging calls: Info (577), Warning (576), Error (577) and Debug (577) use the `Log` call to do the actual work.

See also: Info (577), Warning (576), Error (577), Debug (577), OnGetCustomEvent (581), OnGetCustomEventID (581), OnGetCustomCategory (580)

17.5.11 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 (576), Info (577), Error (577), Debug (577)

17.5.12 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 (576), Info (577), Warning (576), Debug (577)

17.5.13 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 (576), Info (577), Warning (576), Error (577)

17.5.14 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 (576), Warning (576), Error (577), Debug (577)

17.5.15 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 (578), FileName (579)

17.5.16 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 (578), TimeStampFormat (579)

17.5.17 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 (579) property.

See also: FileName (579)

17.5.18 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 (576)

17.5.19 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.

17.5.20 TEventLog.DefaultEventType

Synopsis: Default event type for the Log (576) call.

Declaration: `Property DefaultEventType : TEventType`

Visibility: published

Access: Read,Write

Description: `DefaultEventType` is the event type used by the Log (576) call if it's `EventType` parameter is omitted.

See also: Log (576)

17.5.21 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 (578) 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 (578)

17.5.22 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 (578)

17.5.23 TEventLog.CustomLogType

Synopsis: Custom log type ID.

Declaration: `Property CustomLogType : Word`

Visibility: published

Access: Read,Write

Description: `CustomLogType` is used in the EventTypeToString (574) to format the custom log event type string.

See also: EventTypeToString (574)

17.5.24 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 (574) 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 (574)

17.5.25 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 (581), OnGetCustomEvent (581)

17.5.26 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 (580), OnGetCustomEvent (581)

17.5.27 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 (580), OnGetCustomEventID (581)

17.5.28 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 (578) setting.

See also: LogType (578), TLogMessageEvent (572)

17.5.29 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 (576). Setting it to `False` has the same effect as calling Resume (576).

See also: TEventLog.Pause (576), TEventLog.Resume (576)

Chapter 18

Reference for unit 'ezcgi'

18.1 Used units

Table 18.1: Used units by unit 'ezcgi'

Name	Page
Classes	??
System	??
sysutils	??

18.2 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 (585) or DoGet (585) methods should be overridden.

18.3 Constants, types and variables

18.3.1 Constants

```
hexTable = '0123456789ABCDEF'
```

String constant used to convert a number to a hexadecimal code or back.

18.4 ECGIException

18.4.1 Description

Exception raised by TEZcgi (583).

See also: TEZcgi (583)

18.5 TEZcgi

18.5.1 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 (585) or DoGet (585) methods. To run the program, an instance of this class must be created, and it's Run (584) 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.

18.5.2 Method overview

Page	Method	Description
583	Create	Creates a new instance of the `TEZCGI` component.
583	Destroy	Removes the `TEZCGI` component from memory.
585	DoGet	Method to handle `GET` requests.
585	DoPost	Method to handle `POST` requests.
585	GetValue	Return the value of a request variable.
584	PutLine	Send a line of output to the web-client.
584	Run	Run the CGI application.
584	WriteContent	Writes the content type to standard output.

18.5.3 Property overview

Page	Properties	Access	Description
587	Email	rw	Email of the server administrator.
587	Name	rw	Name of the server administrator.
586	Names	r	Indexed array with available variable names.
585	Values	r	Variables passed to the CGI script.
587	VariableCount	r	Number of available variables.
586	Variables	r	Indexed array with variables as name=value pairs.

18.5.4 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 (586) property.

See also: Variables (586), Names (586), Values (585)

18.5.5 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 (583)

18.5.6 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 (586) property) and calls the DoPost (585) or DoGet (585) 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 (586), DoPost (585), DoGet (585)

18.5.7 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 (585) or DoGet (585) methods.

See also: DoPost (585), DoGet (585), PutLine (584)

18.5.8 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 (584) 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 (584)

18.5.9 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 (585)

18.5.10 TEZcgi.DoPost

Synopsis: Method to handle `POST` requests.

Declaration: `procedure DoPost;  Virtual`

Visibility: public

Description: `DoPost` is called by the Run (584) 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 (584), DoGet (585)

18.5.11 TEZcgi.DoGet

Synopsis: Method to handle `GET` requests.

Declaration: `procedure DoGet;  Virtual`

Visibility: public

Description: `DoGet` is called by the Run (584) 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 (584), DoPost (585)

18.5.12 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_HOSTDNS 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 (586), TEZCGI.Names (586), TEZCGI.GetValue (585), TEZcgi.VariableCount (587)

18.5.13 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 (587). Any other value will result in an exception being raised.

See also: TEZcgi.Variables (586), TEZcgi.Values (585), TEZcgi.GetValue (585), TEZcgi.VariableCount (587)

18.5.14 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 (587). Any other value will result in an exception being raised.

See also: TEZcgi.Names (586), TEZcgi.Values (585), TEZcgi.GetValue (585), TEZcgi.VariableCount (587)

18.5.15 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 (586) and Variables (586) properties.

See also: Names (586), Variables (586), TEZcgi.Values (585), TEZcgi.GetValue (585)

18.5.16 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 (584) method.

See also: TEZcgi.Run (584), TEZcgi.Email (587)

18.5.17 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 (584) method.

See also: TEZcgi.Run (584), TEZcgi.Name (587)

Chapter 19

Reference for unit 'fpjson'

19.1 Used units

Table 19.1: Used units by unit 'fpjson'

Name	Page
Classes	??
Contnrs	213
System	??
sysutils	??
Variants	??

19.2 Overview

The JSON unit implements JSON support for Free Pascal. It contains the data structures (TJSONData (612) and descendent objects) to treat JSON data and output JSON as a string TJSONData.AsJSON (621). The generated JSON can be formatted in several ways TJSONData.FormatJSON (617).

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 (624), TJSONFloatNumber (622) or TJSONInt64Number (623), depending on the type of the number.

Strings in TJSONString (639).

Boolean in TJSONBoolean (611).

null is supported using TJSONNull (625)

Array is supported using TJSONArray (600)

Object is supported using TJSONObject (627)

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 (598) and then the GetJSON (596) 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 (595) 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 (596) and SetJSONInstanceType (597) functions can be used to get or set the classes that must be used. the default parser used by GetJSON (596) will also use these functions.

19.3 Constants, types and variables

19.3.1 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 (617) behave as TJSONData.AsJSON (621) with TJSONData.CompressedJSON equal to True and TJSONData.UnquotedMemberNames equal to True.

```
AsCompressedJSON = [foSingleLineArray, foSingleLineObject, foskipWhiteSpace
  ]
```

AsCompressedJSON can be used to let TJSONData.FormatJSON (617) behave as TJSONData.AsJSON (621) with TJSONData.CompressedJSON (617) equal to True

```
AsJSONFormat = [foSingleLineArray, foSingleLineObject]
```

AsJSONFormat contains the options that make TJSONData.FormatJSON (617) behave like TJSONData.AsJSON (621)

```
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.

19.3.2 Types

```
PJSONCharType = ^TJSONCharType
```

PJSONCharType is a pointer to a TJSONCharType (591) character. It is used while parsing JSON.

```
TFormatOption = (foSingleLineArray,foSingleLineObject,
  foDoNotQuoteMembers,foUseTabchar,foSkipWhiteSpace,
  foSkipWhiteSpaceOnlyLeading)
```

Table 19.2: Enumeration values for type TFormatOption

Value	Explanation
foDoNotQuoteMembers	Do not use quote characters around object member names.
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 (617) function.

```
TFormatOptions = Set of TFormatOption
```

TFormatOptions is the set definition used to specify options in TJSONData.FormatJSON (617).

```
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 (600) class. It is used in CreateJSONArray (595).

```
TJSONArrayIterator = procedure(Item: TJSONData; Data: TObject;
  var Continue: Boolean) of object
```

TJSONArrayIterator is the procedural callback used by TJSONArray.Iterate (602) 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 (611). It is used in the factory methods.

```
TJSONCharType = AnsiChar
```

TJSONCharType is the type of a single character in a TJSONStringType (594) string. It is used by the parser.

```
TJSONDataClass = Class of TJSONData
```

TJSONDataClass is used in the CreateJSON (595), SetJSONInstanceType (597) and GetJSONInstanceType (596) functions to set the actual classes used when creating JSON data.

```
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 (622). It is used in the factory methods.

```
TJSONInstanceType = (jitUnknown,jitNumberInteger,jitNumberInt64,
  jitNumberQWord,jitNumberFloat,jitString,jitBoolean
  ,
                      jitNull,jitArray,jitObject)
```

Table 19.3: 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 (612) descendent to create for a particular data item. It is a more fine-grained division than TJSONType (594)

```
TJSONInt64NumberClass = Class of TJSONInt64Number
```

TJSONInt64NumberClass is the class type of TJSONInt64Number (623). It is used in the factory methods.

```
TJSONIntegerNumberClass = Class of TJSONIntegerNumber
```

TJSONIntegerNumberClass is the class type of TJSONIntegerNumber (624). 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 (593)

```
TJSONNullClass = Class of TJSONNull
```

`TJSONNullClass` is the class type of TJSONNull (625). It is used in the factory methods.

```
TJSONNumberType = (ntFloat,ntInteger,ntInt64,ntQWord)
```

Table 19.4: 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 (627) value.

```
TJSONObjectClass = Class of TJSONObject
```

`TJSONObject` is the class type for the TJSONObject (627) class. It is used in CreateJSONObject (595).

```
TJSONObjectIterator = procedure(const AName: TJSONStringType;
  Item: TJSONData; Data: TObject;
  var Continue: Boolean) of object
```

`TJSONObjectIterator` is the procedural callback used by TJSONObject.Iterate (630) 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 (596) 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 (597) method, the actual types can be retrieved with GetJSONInstanceType (596)

```
TJSONQWordNumberClass = Class of TJSONQWordNumber
```

TJSONQwordNumberClass is the class type of TJSONQWordNumber (638). It is used in the factory methods.

```
TJSONStringClass = Class of TJSONString
```

TJSONStringClass is the class type of TJSONString (639). 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 (612). It is used in the SetJSONStringParserHandler (598) and GetJSONStringParser-Handler (597) 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 19.5: 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 (613) 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 (619) property.

```
TJSONVariant = variant
```

TJSONVariant resolves to Variant on native platforms, JSValue in javascript runtimes.

19.4 Procedures and functions

19.4.1 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 (596) to do so. The classes to be used can be set using the SetJSONInstanceType (597).

The JSON parser uses these functions to create instances of TJSONData (612).

Errors: None.

See also: GetJSONInstanceType (596), SetJSONInstanceType (597), GetJSON (596), CreateJSONArray (595), CreateJSONObject (595)

19.4.2 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 (596), SetJSONInstanceType (597), GetJSON (596), CreateJSON (595), TJSONArray (600)

19.4.3 CreateJSONObject

Synopsis: Create a JSON object.

Declaration:
```
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 (628) 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 (596), SetJSONInstanceType (597), GetJSON (596), CreateJSON (595), TJSONObject (627)

19.4.4 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 (598) 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 (596), SetJSONParserHandler (598), TJSONData (612)

19.4.5 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 (612) 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 (595) 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 (595), TJSONData (612), GetJSON (596)

19.4.6 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 (598), GetJSONStringParserHandler (597), TJSONParserHandler (593), GetJSON (596)

19.4.7 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 (598) call.

See also: SetJSONStringParserHandler (598), SetJSONParserHandler (598), GetJSONParserHandler (596)

19.4.8 JSONStringToString

Synopsis: Convert a JSON-escaped string to a string.

Declaration:
```
function JSONStringToString(const S: TJSONStringType) : 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 (598), JSONTypeName (597)

19.4.9 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 (598), JSONStringToString (597)

19.4.10 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 (612) 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 (595) 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 (596), CreateJSON (595)

19.4.11 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 (598) call.

See also: SetJSONStringParserHandler (598), GetJSONParserHandler (596), TJSONParserHandler (593), GetJSON (596)

19.4.12 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 (598). It sets a callback that will be used by the GetJSON (596) 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 (598) 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 (597), SetJSONParserHandler (598), GetJSONParserHandler (596)

19.4.13 StringToJSONString

Synopsis: Convert a string to a JSON-escaped string.

Declaration:
```
function StringToJSONString(const S: TJSONStringType; 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 (597), JSONTypeName (597)

19.5 TJSONEnum

```
TJSONEnum = record
  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).

19.6 EJSON

19.6.1 Description

EJSON is the exception raised by the JSON implementation to report JSON error.

19.7 TBaseJSONEnumerator

19.7.1 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 (599)

19.7.2 Method overview

Page	Method	Description
599	GetCurrent	Return the current value of the enumerator.
600	MoveNext	Move to next value in array/object.

19.7.3 Property overview

Page	Properties	Access	Description
600	Current	r	Return the current value of the enumerator.

19.7.4 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 (599) value.

See also: TJSONEnum (599)

19.7.5 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 (599), TJSONData (612)

19.7.6 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 (599). It is only valid after MoveNext (600) returned `True`.

See also: TJSONEnum (599), TJSONData (612), MoveNext (600)

19.8 TJSONArray

19.8.1 Description

`TJSONArrayClass` is the class type of TJSONArray (600). It is used in the factory methods.

See also: TJSONArray (600), SetJSONInstanceType (597), GetJSONInstanceType (596)

19.8.2 Method overview

Page	Method	Description
603	Add	Add a JSON value to the array.
603	Clear	Clear the array.
602	Clone	Clone the JSON array.
601	Create	Create a new instance of JSON array data.
604	Delete	Delete an element from the list by index.
602	Destroy	Free the JSON array.
604	Exchange	Exchange 2 elements in the list.
604	Extract	Extract an element from the array.
603	GetEnumerator	Get an array enumerator.
603	IndexOf	Return index of JSONData instance in array.
605	Insert	Insert an element in the array.
602	Iterate	Iterate over all elements in the array.
602	JSONType	native JSON data type.
605	Move	Move a value from one location to another.
605	Remove	Remove an element from the list.
606	Sort	Sort the items in the array.

19.8.3 Property overview

Page	Properties	Access	Description
610	Arrays	rw	Get or set elements as JSON array values.
610	Booleans	rw	Get or set elements as boolean values.
609	Floats	rw	Get or set elements as floating-point numerical values.
607	Int64s	rw	Get or set elements as Int64 values.
607	Integers	rw	Get or set elements as integer values.
606	Items		Indexed access to the values in the array.
608	LargeInts	rw	Get or set elements as LargeInt values.
606	Nulls	r	Check which elements are null.
610	Objects	rw	Get or set elements as JSON object values.
608	QWords	rw	Get or set elements as QWord values.
609	Strings	rw	Get or set elements as string values.
606	Types	r	JSON types of elements in the array.
608	UnicodeStrings	rw	Get or set elements as Unicode string values.

19.8.4 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 (612) 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 (596)

19.8.5 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 (600) instance.

See also: TJSONArray.Clear (603), TJSONArray.Create (601)

19.8.6 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 (613)

19.8.7 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 (617)

19.8.8 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 (591)

19.8.9 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 (603)

19.8.10 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 (599).

See also: TJSONEnum (599), TJSONData.GetEnumerator (614)

19.8.11 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 (618) returns 0.

See also: Delete (604), Extract (604)

19.8.12 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: string) : 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 (612) 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 (604), Extract (604)

19.8.13 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 (612) element is freed.

Errors: If an invalid index is passed, an exception is raised.

See also: Clear (603), Add (603), Extract (604), Exchange (604)

19.8.14 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.

19.8.15 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 (604) does. In difference with `Delete`, it does not free the object instance. Instead, it returns the extracted element.

See also: Delete (604), Clear (603), Insert (605), Add (603)

19.8.16 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: string)
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 (603), it converts plain pascal values to JSON values.

Note that when inserting a TJSONData (612) 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 (603), Delete (604), Extract (604), Clear (603)

19.8.17 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 (604), Extract (604), Insert (605)

19.8.18 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 (604), Extract (604)

19.8.19 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 (**??**)

19.8.20 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 (618). `TJSONArray` simply declares it as the default property.

See also: TJSONData.Items (618)

19.8.21 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 (613) result of the elements in the array. Accessing it is equivalent to accessing

```
Items[Index].JSONType
```

See also: TJSONData.JSONType (613), TJSONData.Items (618)

19.8.22 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 (621) property when reading. It is then equivalent to accessing

```
Items[Index].IsNull
```

See also: TJSONData.JSONType (613), TJSONData.Items (618), TJSONData.IsNull (621), TJSONArray.Types (606)

19.8.23 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 (620) 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 (618), TJSONData.IsNull (621), TJSONArray.Types (606), TJSONArray.Int64s (607), TJSONArray.QWords (608), TJSONArray.Floats (609), TJSONArray.Strings (609), TJSONArray.Booleans (610)

19.8.24 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 (619) 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 (618), TJSONData.IsNull (621), TJSONArray.Types (606), TJSONArray.Integers (607), TJSONArray.Floats (609), TJSONArray.Strings (609), TJSONArray.Booleans (610), TJSONArray.QWords (608)

19.8.25 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 (620) 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 (620), TJSONData.Items (618), TJSONData.IsNull (621), TJSONArray.Types (606), TJSONArray.Integers (607), TJSONArray.Floats (609), TJSONArray.Strings (609), TJSONArray.Booleans (610), TJSONArray.QWords (608)

19.8.26 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 (619) 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 (588), AsQWord (619), IsNull (621), Types (606), Integers (607), Floats (609), Strings (609), Booleans (610)

19.8.27 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 (619) 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 (618), TJSONData.IsNull (621), TJSONArray.Types (606), TJSONArray.Integers (607), TJSONArray.QWords (608), TJSONArray.Floats (609), TJSONArray.Int64s (607), TJSONArray.Booleans (610), TJSONArray.Strings (609)

19.8.28 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 (618) 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 (618), TJSONData.IsNull (621), TJSONArray.Types (606), TJSONArray.Integers (607), TJSONArray.QWords (608), TJSONArray.Floats (609), TJSONArray.Int64s (607), TJSONArray.Booleans (610)

19.8.29 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 (620) 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 (618), TJSONData.IsNull (621), TJSONArray.Types (606), TJSONArray.Integers (607), TJSONArray.Strings (609), TJSONArray.Int64s (607), TJSONArray.QWords (608), TJSONArray.Booleans (610)

19.8.30 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 (621) 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 (618), TJSONData.IsNull (621), TJSONArray.Types (606), TJSONArray.Integers (607), TJSONArray.Strings (609), TJSONArray.Int64s (607), TJSONArray.QWords (608), TJSONArray.Floats (609)

19.8.31 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 (618), TJSONData.IsNull (621), TJSONArray.Types (606), TJSONArray.Integers (607), TJSONArray.Strings (609), TJSONArray.Int64s (607), TJSONArray.QWords (608), TJSONArray.Floats (609), TJSONArray.Objects (610)

19.8.32 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 (618), TJSONData.IsNull (621), TJSONArray.Types (606), TJSONArray.Integers (607), TJSONArray.Strings (609), TJSONArray.Int64s (607), TJSONArray.QWords (608), TJSONArray.Floats (609), TJSONArray.Arrays (610)

19.9 TJSONBoolean

19.9.1 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 (622), TJSONIntegerNumber (624), TJSONInt64Number (623), TJSONBoolean (611), TJSONNull (625), TJSONArray (600), TJSONObject (627)

19.9.2 Method overview

Page	Method	Description
612	Clear	Clear data.
612	Clone	Clone boolean value.
611	Create	Create a new instance of boolean JSON data.
611	JSONType	native JSON data type.

19.9.3 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 (625), TJSONFloatNumber.Create (622), TJSONInt64Number.Create (623), TJSONString.Create (639), TJSONArray.Create (601), TJSONObject.Create (628)

19.9.4 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 (613)

19.9.5 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 (614)

19.9.6 TJSONBoolean.Clone

Synopsis: Clone boolean value.

Declaration: `function Clone : TJSONData;  Override`

Visibility: public

Description: `Clone` overrides TJSONData.Clone (617) and creates an instance of the same class with the same boolean value.

See also: TJSONData.Clone (617)

19.10 TJSONData

19.10.1 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 (624), TJSONFloatNumber (622) or TJSONInt64Number (623), depending on the type of the number.

Strings can be represented with TJSONString (639).

Boolean can be represented with TJSONBoolean (611).

null is supported using TJSONNull (625)

Array data can be represented using TJSONArray (600)

Object data can be supported using TJSONObject (627)

See also: TJSONIntegerNumber (624), TJSONString (639), TJSONBoolean (611), TJSONNull (625), TJSONArray (600), TJSONObject (627)

19.10.2 Method overview

Page	Method	Description
614	Clear	Clear the raw value of this data object.
617	Clone	Duplicate the value of the JSON data.
613	Create	Create a new instance of TJSONData.
614	DumpJSON	Fast, memory efficient dump of JSON in stream.
614	FindPath	Find data by name.
617	FormatJSON	Return a formatted JSON representation of the data.
614	GetEnumerator	Return an enumerator for the data.
616	GetPath	Get data by name.
613	JSONType	The native JSON data type represented by this object.

19.10.3 Property overview

Page	Properties	Access	Description
621	AsBoolean	rw	Access the raw JSON value as a boolean.
620	AsFloat	rw	Access the raw JSON value as a float.
619	AsInt64	rw	Access the raw JSON value as an 64-bit integer.
620	AsInteger	rw	Access the raw JSON value as an 32-bit integer.
621	AsJSON	r	Return a JSON representation of the value.
620	AsLargeInt	rw	Access to data as largeint.
619	AsQWord	rw	Access the raw JSON value as an 64-bit unsigned integer.
618	AsString	rw	Access the raw JSON value as a string.
619	AsUnicodeString	rw	Return the value as a Unicode string.
617	CompressedJSON	rw	Compress JSON - skip whitespace.
618	Count	r	Number of sub-items for this data element.
621	IsNull	r	Is the data a null value ?
618	Items	rw	Indexed access to sub-items.
618	Value	rw	The value of this data object as a variant.

19.10.4 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 (594)

19.10.5 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 (625), TJSONString.Create (639), TJSONBoolean.Create (611), TJSONArray.Create (601), TJSONObject.Create (628)

19.10.6 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 (613)

19.10.7 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 (621) and writing it to stream.

Errors: None.

See also: TJSONData.FormatJSON (617), TJSONData.AsJSON (621)

19.10.8 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 (612) type, but a TJSONEnum (599) structure, which contains the value, and for structured types, the key (numerical or string).

See also: TJSONEnum (599), TJSONArray (600), TJSONObject (627)

19.10.9 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 (627), a member can be specified using its name, and the object value itself can be retrieved with the empty path.
- For JSON Arrays (627), 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 (600), TJSONObject (627), GetPath (616)

19.10.10 TJSONData.GetPath

Synopsis: Get data by name.

Declaration: `function GetPath(const APath: TJSONStringType) : TJSONData`

Visibility: public

Description: `GetPath` is identical to FindPath (614) but raises an exception if no element was found. The exception message contains the piece of path that was not found.

Errors: An EJSON (599) exception is raised if the path does not exist.

See also: FindPath (614)

19.10.11 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 (614), EOutOfMemory (**??**)

19.10.12 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 (588) property, but for complex values (TJSONArray (600) and TJSONObject (627)) 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 (621), TFormatOptions (591)

19.10.13 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 (621) 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 (617), TJSONData.AsJSON (621), TJSONData.UnquotedMemberNames (612), TJSONData.AsCompressedJSON (612), TJSONObject.UnquotedMemberNames (633)

19.10.14 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 (618)

19.10.15 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 (600) and TJSONObject (627)) override this method to return the `Index`-th element in the list.

See also: Count (618), TJSONArray (600), TJSONObject (627)

19.10.16 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 (594)), 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 (594)

19.10.17 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 (639) this will return the native value.

For complex values, reading or writing this property will result in an `EConvertError` exception.

See also: AsInteger (620), Value (618), AsInt64 (619), AsFloat (620), AsBoolean (621), IsNull (621), AsJSON (621)

19.10.18 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 (618)

19.10.19 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 (623) this will return the native value.

For complex values, reading or writing this property will always result in an `EConvertError` exception.

See also: AsFloat (620), Value (618), AsInteger (620), AsString (618), AsBoolean (621), IsNull (621), AsJSON (621)

19.10.20 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 (638) this will return the native value.

For complex values, reading or writing this property will always result in an `EConvertError` exception.

See also: AsFloat (620), Value (618), AsInteger (620), AsInt64 (619), AsString (618), AsBoolean (621), IsNull (621), AsJSON (621)

19.10.21 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.

19.10.22 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 (622) this will return the native value.

For complex values, reading or writing this property will always result in an `EConvertError` exception.

See also: AsInteger (620), Value (618), AsInt64 (619), AsString (618), AsBoolean (621), IsNull (621), AsJSON (621)

19.10.23 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 (624) this will return the native value.

For complex values, reading or writing this property will always result in an `EConvertError` exception.

See also: AsFloat (620), Value (618), AsInt64 (619), AsString (618), AsBoolean (621), IsNull (621), AsJSON (621)

19.10.24 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 (611) this will return the native value.

For complex values, reading or writing this property will always result in an `EConvertError` exception.

See also: AsFloat (620), Value (618), AsInt64 (619), AsString (618), AsInteger (620), IsNull (621), AsJSON (621)

19.10.25 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 (625) instance. In all other cases, it is `False`. This value cannot be set.

See also: TJSONType (594), JSONType (613), TJSONNull (625), AsFloat (620), Value (618), AsInt64 (619), AsString (618), AsInteger (620), AsBoolean (621), AsJSON (621)

19.10.26 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 (620), Value (618), AsInt64 (619), AsString (618), AsInteger (620), AsBoolean (621), AsJSON (621)

19.11 TJSONFloatNumber

19.11.1 Description

`TJSONFloatNumber` must be used whenever floating point data must be represented. It can handle TJSONFloat (592) data (normally a double). For integer data, TJSONIntegerNumber (624) or TJSONInt64Number (623) are better suited.

See also: TJSONNumber (626), TJSONFloat (592), TJSONIntegerNumber (624), TJSONInt64Number (623)

19.11.2 Method overview

Page	Method	Description
623	Clear	Clear value.
623	Clone	Clone floating point value.
622	Create	Create a new floating-point value.
622	NumberType	Kind of numerical data managed by this class.

19.11.3 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 (625), TJSONInt64Number.Create (623)

19.11.4 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 (593), TJSONData.JSONtype (613)

19.11.5 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 (614)

19.11.6 TJSONFloatNumber.Clone

Synopsis: Clone floating point value.

Declaration: `function Clone : TJSONData;  Override`

Visibility: public

Description: `Clone` overrides TJSONData.Clone (617) and creates an instance of the same class with the same floating-point value.

See also: TJSONData.Clone (617)

19.12 TJSONInt64Number

19.12.1 Description

`TJSONInt64Number` must be used whenever 64-bit integer data must be represented. For 32-bit integer data, TJSONIntegerNumber (624) must be used.

See also: TJSONFloatNumber (622), TJSONIntegerNumber (624)

19.12.2 Method overview

Page	Method	Description
624	Clear	Clear value.
624	Clone	Clone 64-bit integer value.
623	Create	Create a new instance of 64-bit integer JSON data.
624	NumberType	Kind of numerical data managed by this class.

19.12.3 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 (625), TJSONFloatNumber.Create (622)

19.12.4 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 (593), TJSONData.JSONtype (613)

19.12.5 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 (614)

19.12.6 TJSONInt64Number.Clone

Synopsis: Clone 64-bit integer value.

Declaration: `function Clone : TJSONData; Override`

Visibility: public

Description: `Clone` overrides TJSONData.Clone (617) and creates an instance of the same class with the same 64-bit integer value.

See also: TJSONData.Clone (617)

19.13 TJSONIntegerNumber

19.13.1 Description

`TJSONIntegerNumber` must be used whenever 32-bit integer data must be represented. For 64-bit integer data, TJSONInt64Number (623) must be used.

See also: TJSONFloatNumber (622), TJSONInt64Number (623)

19.13.2 Method overview

Page	Method	Description
625	Clear	Clear value.
625	Clone	Clone 32-bit integer value.
625	Create	Create a new instance of 32-bit integer JSON data.
625	NumberType	Kind of numerical data managed by this class.

19.13.3 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 (622), TJSONInt64Number.Create (623)

19.13.4 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 (593), TJSONData.JSONtype (613)

19.13.5 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 (614)

19.13.6 TJSONIntegerNumber.Clone

Synopsis: Clone 32-bit integer value.

Declaration: `function Clone : TJSONData;  Override`

Visibility: public

Description: `Clone` overrides TJSONData.Clone (617) and creates an instance of the same class with the same 32-bit integer value.

See also: TJSONData.Clone (617)

19.14 TJSONNull

19.14.1 Description

`TJSONNull` must be used whenever a `null` value must be represented.

See also: TJSONFloatNumber (622), TJSONIntegerNumber (624), TJSONInt64Number (623), TJSONBoolean (611), TJSONString (639), TJSONArray (600), TJSONObject (627)

19.14.2 Method overview

Page	Method	Description
626	Clear	Clear data.
626	Clone	Clone boolean value.
626	JSONType	native JSON data type.

19.14.3 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 (613)

19.14.4 TJSONNull.Clear

Synopsis: Clear data.

Declaration: `procedure Clear;  Override`

Visibility: public

Description: `Clear` does nothing.

See also: TJSONData.Clear (614)

19.14.5 TJSONNull.Clone

Synopsis: Clone boolean value.

Declaration: `function Clone : TJSONData;  Override`

Visibility: public

Description: `Clone` overrides TJSONData.Clone (617) and creates an instance of the same class.

See also: TJSONData.Clone (617)

19.15 TJSONNumber

19.15.1 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 (624), TJSONInt64Number (623) or TJSONFloatNumber (622) should be instantiated.

See also: TJSONIntegerNumber (624), TJSONInt64Number (623), TJSONFloatNumber (622)

19.15.2 Method overview

Page	Method	Description
627	JSONType	native JSON data type.
627	NumberType	Kind of numerical data managed by this class.

19.15.3 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 (613)

19.15.4 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 (624), TJSONInt64Number (623), TJSONFloatNumber (622), JSONType (588)

19.16 TJSONObject

19.16.1 Description

`TJSONObjectClass` is the class type of TJSONObject (627). It is used in the factory methods.

See also: TJSONObject (627), SetJSONInstanceType (597), GetJSONInstanceType (596)

19.16.2 Method overview

Page	Method	Description
632	Add	Add a name, value to the object.
632	Clear	Clear the object.
629	Clone	Clone the JSON object.
628	Create	Create a new instance of JSON object data.
632	Delete	Delete an element from the list by index.
629	Destroy	Free the JSON object.
633	Extract	Extract an element from the object.
631	Find	Find an element by name.
631	Get	Retrieve a value by name.
630	GetEnumerator	Get an object enumerator.
630	IndexOf	Return index of JSONData instance in object.
630	IndexOfName	Return index of name in item list.
630	Iterate	Iterate over all elements in the object.
629	JSONType	native JSON data type.
633	Remove	Remove item by instance.

19.16.3 Property overview

Page	Properties	Access	Description
637	Arrays	rw	Named access to JSON array values.
637	Booleans	rw	Named access to boolean values.
634	Elements	rw	Name-based access to JSON values in the object.
635	Floats	rw	Named access to float values.
635	Int64s	rw	Named access to int64 values.
635	Integers	rw	Named access to integer values.
636	LargeInts	rw	Get or set elements as LargeInt values.
634	Names	r	Indexed access to the names of elements.
635	Nulls	rw	Named access to `null` values.
638	Objects	rw	Named access to JSON object values.
636	QWords	rw	Named access to QWord values.
637	Strings	rw	Named access to string values.
634	Types	r	Types of values in the object.
636	UnicodeStrings	rw	Named access to Unicode string values.
633	UnquotedMemberNames	rw	Should member names be written unquoted or quoted in JSON.

19.16.4 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 (612) 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 (632), GetJSONInstanceType (596)

19.16.5 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 (627) instance.

See also: TJSONObject.Clear (632), TJSONObject.Create (628)

19.16.6 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 (613)

19.16.7 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 (617)

19.16.8 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 (599).

See also: TJSONEnum (599), TJSONData.GetEnumerator (614)

19.16.9 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 (593)

19.16.10 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 (618) 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 (632), IndexOfName (630)

19.16.11 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 (618) 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 (630)

19.16.12 TJSONObject.Find

Synopsis: Find an element by name.

Declaration:
```
function Find(const AName: string) : TJSONData;  Overload
function Find(const AName: string; 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 (630), IndexOfName (630)

19.16.13 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 (635), Int64s (635), Booleans (637), Strings (637), Floats (635), Arrays (637), and Objects (638) will raise an exception if the name is not found. The `Get` function does not raise an exception.

See also: Integers (635), Int64s (635), Booleans (637), Strings (637), Floats (635), Arrays (637), Objects (638)

19.16.14 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 (618) returns 0.

See also: Delete (632), Extract (633)

19.16.15 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 (612) 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 (635), Int64s (635), Booleans (637), Strings (637), Floats (635), Arrays (637) and Objects (638) 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 (635), Int64s (635), Booleans (637), Strings (637), Floats (635), Arrays (637), Objects (638)

19.16.16 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 (612) 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 (632), Add (632), Extract (633), Exchange (627)

19.16.17 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 (632), Extract (633)

19.16.18 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 (588) 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 (588), Clear (588), Insert (588), Add (588)

19.16.19 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 (617)

19.16.20 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 (634), Elements (634)

19.16.21 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 (599) exception is raised.

See also: Items (618), Names (634), Types (634), Integers (635), Int64s (635), Booleans (637), Strings (637), Floats (635), Arrays (637), Objects (638)

19.16.22 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 (599) exception is raised.

See also: Items (618), Names (634), Elements (634), Integers (635), Int64s (635), Booleans (637), Strings (637), Floats (635), Arrays (637), Nulls (635), Objects (638)

19.16.23 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 (599) exception is raised when reading. When writing, any existing value is replaced by a `null` value.

See also: TJSONData.Items (618), TJSONObject.Names (634), Elements (588), Integers (588), Int64s (588), Booleans (588), Strings (588), Floats (588), Arrays (588), Types (588), Objects (588)

19.16.24 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 (599) exception is raised when reading. When writing, any existing value is replaced by the specified floating-point value.

See also: Items (618), Names (634), Elements (634), Integers (635), Int64s (635), Booleans (637), Strings (637), Nulls (635), Arrays (637), Types (634), Objects (638)

19.16.25 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 (599) exception is raised when reading. When writing, any existing value is replaced by the specified integer value.

See also: TJSONData.Items (618), Names (634), Elements (634), Floats (635), Int64s (635), Booleans (637), Strings (637), Nulls (635), Arrays (637), Types (634), Objects (638)

19.16.26 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 (599) exception is raised when reading. When writing, any existing value is replaced by the specified int64 value.

See also: Items (618), Names (634), Elements (634), Floats (635), Integers (635), Booleans (637), Strings (637), Nulls (635), Arrays (637), Types (634), Objects (638)

19.16.27 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 (599) exception is raised when reading. When writing, any existing value is replaced by the specified QWord value.

See also: TJSONData.Items (618), Names (634), Elements (634), Floats (635), Integers (635), Booleans (637), Strings (637), Nulls (635), Arrays (637), Types (634), Objects (638)

19.16.28 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 (620) 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 (620), TJSONData.Items (618), TJSONData.IsNull (621), TJSONObject.Types (634), TJSONObject.Integers (635), TJSONObject.Floats (635), TJSONObject.Strings (637), TJSONObject.Booleans (637), TJSONObject.QWords (636)

19.16.29 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 (599) exception is raised when reading. When writing, any existing value is replaced by the specified UnicodeString value.

See also: Items (618), Names (634), Elements (634), Floats (635), Integers (635), Booleans (637), Int64s (635), Nulls (635), Arrays (637), Types (634), Objects (638), Strings (637)

19.16.30 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 (599) exception is raised when reading. When writing, any existing value is replaced by the specified string value.

See also: Items (618), Names (634), Elements (634), Floats (635), Integers (635), Booleans (637), Int64s (635), Nulls (635), Arrays (637), Types (634), Objects (638), UnicodeStrings (636)

19.16.31 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 (599) exception is raised when reading. When writing, any existing value is replaced by the specified boolean value.

See also: Items (618), Names (634), Elements (634), Floats (635), Integers (635), Strings (637), Int64s (635), Nulls (635), Arrays (637), Types (634), Objects (638)

19.16.32 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 (599) exception is raised when reading. When writing, any existing value is replaced by the specified JSON array.

See also: Items (618), Names (634), Elements (634), Floats (635), Integers (635), Strings (637), Int64s (635), Nulls (635), Booleans (637), Types (634), Objects (638)

19.16.33 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 (599) exception is raised when reading. When writing, any existing value is replaced by the specified JSON object.

See also: TJSONData.Items (618), Names (634), Elements (634), Floats (635), Integers (635), Strings (637), Int64s (635), Nulls (635), Booleans (637), Types (634), Arrays (637)

19.17 TJSONQWordNumber

19.17.1 Description

`TJSONQWordNumber` must be used whenever 64-bit unsigned integer data must be represented. For 32-bit integer data, TJSONIntegerNumber (624) must be used.For 64-bit signed integer data, TJSONInt64Number (623) must be used.

See also: TJSONFloatNumber (622), TJSONIntegerNumber (624), TJSONInt64Number (623)

19.17.2 Method overview

Page	Method	Description
639	Clear	Clear value.
639	Clone	Clone 64-bit unsigned integer value.
638	Create	Create a new instance of 64-bit unsigned integer JSON data.
638	NumberType	Kind of numerical data managed by this class.

19.17.3 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 (625), TJSONInt64Number.Create (623), TJSONFloatNumber.Create (622)

19.17.4 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 (593), TJSONData.JSONtype (613)

19.17.5 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 (614)

19.17.6 TJSONQWordNumber.Clone

Synopsis: Clone 64-bit unsigned integer value.

Declaration: `function Clone : TJSONData;  Override`

Visibility: public

Description: `Clone` overrides TJSONData.Clone (617) and creates an instance of the same class with the 64-bit unsigned integer value.

See also: TJSONData.Clone (617)

19.18 TJSONString

19.18.1 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 (622), TJSONIntegerNumber (624), TJSONInt64Number (623), TJSONBoolean (611), TJSONNull (625), TJSONArray (600), TJSONObject (627)

19.18.2 Method overview

Page	Method	Description
640	Clear	Clear value.
640	Clone	Clone string value.
639	Create	Create a new instance of string JSON data.
640	JSONType	native JSON data type.

19.18.3 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 (625), TJSONFloatNumber.Create (622), TJSONInt64Number.Create (623), TJSONBoolean.Create (611), TJSONArray.Create (601), TJSONObject.Create (628)

19.18.4 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 (613)

19.18.5 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 (614)

19.18.6 TJSONString.Clone

Synopsis: Clone string value.

Declaration: `function Clone : TJSONData;  Override`

Visibility: public

Description: `Clone` overrides TJSONData.Clone (617) and creates an instance of the same class with the same string value.

See also: TJSONData.Clone (617)

Chapter 20

Reference for unit 'fpmimetypes'

20.1 Used units

Table 20.1: Used units by unit 'fpmimetypes'

Name	Page
Classes	??
Contnrs	213
System	??
sysutils	??

20.2 Overview

The fpmimetypes unit contains a class which handles mapping of filename extensions to MIME (Multipurpose Internet Mail Extensions) types. The TFPMimeTypes (642) class handles this mapping. A global instance of this class is available through the MimeTypes (641) function. The list of known mime types can be instantiated through the LoadKnownTypes (643) method, or a file in the standard mime.types format can be loaded through the LoadFromFile (644) method.

20.3 Procedures and functions

20.3.1 MimeTypes

Synopsis: Global `TFPMimeTypes` instance.

Declaration: `function MimeTypes : TFPMimeTypes`

Visibility: default

Description: `MimeTypes` returns a global instance of the TFPMimeTypes (642) class. It is not initialized with a list of extensions, so this instance must still be explicitly initialized with TFPMimeTypes.LoadKnownTypes (643)

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 (642), TFPMimeTypes.LoadKnownTypes (643)

20.4 TFPMimeTypes

20.4.1 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 (643) method, or a file in the standard mime.types format can be loaded through the LoadFromFile (644) method.

The associated mime type of a file extension can be retrieved with TFPMimeTypes.GetMimeType (644).

See also: TFPMimeTypes.LoadKnownTypes (643), TFPMimeTypes.LoadFromFile (644), TFPMimeTypes.GetMimeType (644)

20.4.2 Method overview

Page	Method	Description
644	AddType	Add a MIME type to the list.
643	Clear	Clear the list of known `MIME` types.
642	Create	Create a new instance of the `TFPMimeTypes` class.
642	Destroy	Remove instance from memory.
645	GetKnownExtensions	Get a lis of all known extensions.
645	GetKnownMimeTypes	Get a list of all known MIME types.
644	GetMimeExtensions	Get the extensions associated with a MIME type.
644	GetMimeType	Get MIME type of an extension.
643	GetNextExtension	Extract an extension from an extension list as returned by GetMimeExtensions.
644	LoadFromFile	Load mime types from a file in mime.types file format.
643	LoadFromStream	Load mime types from a stream in mime.types file format.
643	LoadKnownTypes	Queries the OS for a list of known MIME types.

20.4.3 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 (642)

20.4.4 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 (642)

20.4.5 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 (643)

20.4.6 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 (644)

20.4.7 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 (644)

20.4.8 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 (644)

20.4.9 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 (643)

20.4.10 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 (644), TFPMimeTypes.GetMimeType (644)

20.4.11 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 (644)

20.4.12 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 (644)

20.4.13 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 (645)

20.4.14 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 (645)

20.5 TMimeType

20.5.1 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 (642) class.

See also: TMimeType.MimeType (646), TMimeType.Extensions (646), TFPMimeTypes (642)

20.5.2 Method overview

Page	Method	Description
646	Create	Create a new instance of `TMimeType`.
646	MergeExtensions	Merge extensions in the list of extensions.

20.5.3 Property overview

Page	Properties	Access	Description
646	Extensions	rw	Semicolon-separated list of extensions associated with `MimeType`.
646	MimeType	rw	Mime type name.

20.5.4 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 (646)TMimeType.Extensions (646) properties to `aMimeType` and `aExtensions`.

See also: TMimeType.MimeType (646), TMimeType.Extensions (646)

20.5.5 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 (646) in such a way that there are no duplicates.

See also: TMimeType.Extensions (646)

20.5.6 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 (646)

20.5.7 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 (646)

See also: TMimeType.MimeType (646)

Chapter 21

Reference for unit 'fptimer'

21.1 Used units

Table 21.1: Used units by unit 'fptimer'

Name	Page
Classes	??
System	??

21.2 Overview

The fpTimer unit implements a timer class TFPTimer (649) which can be used on all supported platforms. The timer class uses a driver class TFPTimerDriver (651) 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 (647) variable to the class pointer of the driver class. The driver class should descend from TFPTimerDriver (651).

21.3 Constants, types and variables

21.3.1 Types

```
TFPTimerDriverClass = Class of TFPTimerDriver
```

`TFPTimerDriverClass` is the class pointer of TFPTimerDriver (651) it exists mainly for the purpose of being able to set DefaultTimerDriverClass (647), so a custom timer driver can be used for the timer instances.

21.3.2 Variables

```
DefaultTimerDriverClass : TFPTimerDriverClass = Nil
```

DefaultTimerDriverClass contains the TFPTimerDriver (651) class pointer that should be used when a new instance of TFPCustomTimer (648) 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.

21.4 TFPCustomTimer

21.4.1 Description

TFPCustomTimer is the timer class containing the timer's implementation. It relies on an extra driver instance (of type TFPTimerDriver (651)) to do the actual work.

TFPCustomTimer publishes no events or properties, so it is unsuitable for handling in an IDE. The TFPTimer (649) descendent class publishes all needed events of TFPCustomTimer.

See also: TFPTimerDriver (651), TFPTimer (649)

21.4.2 Method overview

Page	Method	Description
648	Create	Create a new timer.
648	Destroy	Release a timer instance from memory.
649	StartTimer	Start the timer.
649	StopTimer	Stop the timer.

21.4.3 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 (648)

21.4.4 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 (648)

21.4.5 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 (649) event is called.

See also: StopTimer (649), Enabled (649), OnTimer (650)

21.4.6 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 (649), Enabled (649), OnTimer (650)

21.5 TFPTimer

21.5.1 Description

`TFPTimer` implements no new events or properties, but merely publishes events and properties already implemented in TFPCustomTimer (648): Enabled (649), OnTimer (650) and Interval (650).

The `TFPTimer` class is suitable for use in an IDE.

See also: TFPCustomTimer (648), Enabled (649), OnTimer (650), Interval (650)

21.5.2 Property overview

Page	Properties	Access	Description
649	Enabled		Start or stop the timer.
650	Interval		Timer tick interval in milliseconds.
650	OnStartTimer		
651	OnStopTimer		
650	OnTimer		Event called on each timer tick.
650	UseTimerThread		

21.5.3 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 (649)), setting it to `False` will stop the timer (calling StopTimer (649)).

See also: StartTimer (649), StopTimer (649), OnTimer (650), Interval (650)

21.5.4 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 (650) 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 (650), Enabled (649)

21.5.5 TFPTimer.UseTimerThread

Declaration: `Property UseTimerThread :`

Visibility: published

Access:

21.5.6 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 (650), Enabled (649)

21.5.7 TFPTimer.OnStartTimer

Declaration: `Property OnStartTimer :`

Visibility: published

Access:

21.5.8 TFPTimer.OnStopTimer

Declaration: `Property OnStopTimer :`

Visibility: published

Access:

21.6 TFPTimerDriver

21.6.1 Description

`TFPTimerDriver` is the abstract timer driver class: it simply provides an interface for the TFPCustomTimer (648) class to use.

The `fpTimer` unit implements a descendent of this class which implements the default timer mechanism.

See also: TFPCustomTimer (648), DefaultTimerDriverClass (647)

21.6.2 Method overview

Page	Method	Description
651	Create	Creates a new driver instance.
651	StartTimer	Start the timer.
652	StopTimer	Stop the timer.

21.6.3 Property overview

Page	Properties	Access	Description
652	Timer	r	Timer tick.
652	TimerStarted	r	True when the timer driver has called its StartTimer method.

21.6.4 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 (652)) a reference to the `ATimer` instance which created the driver instance.

See also: Timer (652), TFPTimer (649)

21.6.5 TFPTimerDriver.StartTimer

Synopsis: Start the timer.

Declaration: `procedure StartTimer; Virtual; Abstract`

Visibility: public

Description: `StartTimer` is called by TFPCustomTimer.StartTimer (649). It should be overridden by descendents of `TFPTimerDriver` to actually start the timer.

See also: TFPCustomTimer.StartTimer (649), TFPTimerDriver.StopTimer (652)

21.6.6 TFPTimerDriver.StopTimer

Synopsis: Stop the timer.

Declaration: `procedure StopTimer; Virtual; Abstract`

Visibility: public

Description: `StopTimer` is called by TFPCustomTimer.StopTimer (649). It should be overridden by descendents of `TFPTimerDriver` to actually stop the timer.

See also: TFPCustomTimer.StopTimer (649), TFPTimerDriver.StartTimer (651)

21.6.7 TFPTimerDriver.Timer

Synopsis: Timer tick.

Declaration: `Property Timer : TFPCustomTimer`

Visibility: public

Access: Read

Description: `Timer` calls the TFPCustomTimer (648) timer event. Descendents of `TFPTimerDriver` should call `Timer` whenever a timer tick occurs.

See also: TFPTimer.OnTimer (650), TFPTimerDriver.StartTimer (651), TFPTimerDriver.StopTimer (652)

21.6.8 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 22

Reference for unit 'gettext'

22.1 Used units

Table 22.1: Used units by unit 'gettext'

Name	Page
Classes	??
System	??
sysutils	??

22.2 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 (656)) 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.

22.3 Constants, types and variables

22.3.1 Constants

```
MOFileHeaderMagic = $950412DE
```

This constant is found as the first integer in a .mo

22.3.2 Types

```
PLongWordArray = ^TLongWordArray
```

Pointer to a TLongWordArray (654) array.

```
PMOStringTable = ^TMOStringTable
```

Pointer to a TMOStringTable (654) array.

```
PPCharArray = ^TPCharArray
```

Pointer to a TPCharArray (654) array.

```
TLongWordArray = Array[0..(1shl30)divSizeOf(LongWord)] of
  LongWord
```

`TLongWordArray` is an array used to define the PLongWordArray (653) 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 (655) records. It should never be used directly, as it would occupy too much memory.

```
TPCharArray = Array[0..(1shl30)divSizeOf(PChar)] of PChar
```

`TLongWordArray` is an array used to define the PPCharArray (654) 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.

22.4 Procedures and functions

22.4.1 GetLanguageIDs

Synopsis: Return the current language IDs.

Declaration:
```
procedure GetLanguageIDs(var Lang: string; var FallbackLang: string)
```

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.

22.4.2 TranslateResourceStrings

Synopsis: Translate the resource strings of the application.

Declaration:
```
procedure TranslateResourceStrings(AFile: TMOFile)
procedure TranslateResourceStrings(const AFilename: string)
```

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 (656) 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 (655), TMOFile (656)

22.4.3 TranslateUnitResourceStrings

Synopsis: Translate the resource strings of a unit.

Declaration:
```
procedure TranslateUnitResourceStrings(const AUnitName: string;
                                       AFile: TMOFile)
procedure TranslateUnitResourceStrings(const AUnitName: string;
                                       const AFilename: string)
```

Visibility: default

Description: `TranslateUnitResourceStrings` is identical in function to TranslateResourceStrings (654), 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 (654), TMOFile (656)

22.5 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.

22.6 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 (655) record at the start of the file.

22.7 EMOFileError

22.7.1 Description

`EMOFileError` is raised in case an TMOFile (656) instance is created with an invalid .mo.

See also: TMOFile (656)

22.8 TMOFile

22.8.1 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.

22.8.2 Method overview

Page	Method	Description
656	Create	Create a new instance of the `TMOFile` class.
656	Destroy	Removes the `TMOFile` instance from memory.
656	Translate	Translate a string.

22.8.3 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 (655) structure, then an EMOFileError (655) exception is raised.

See also: TMOFile.Destroy (656)

22.8.4 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 (656)

22.8.5 TMOFile.Translate

Synopsis: Translate a string.

Declaration:
```
function Translate(AOrig: PChar; ALen: Integer; AHash: LongWord)
                        : string
function Translate(AOrig: string; AHash: LongWord) : string
function Translate(AOrig: string) : string
```

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 23

Reference for unit 'IBConnection'

23.1 Used units

Table 23.1: Used units by unit 'IBConnection'

Name	Page
BufDataset	142
Classes	??
DB	351
dbconst	??
ibase60dyn	??
SQLDB	844
System	??
sysutils	??

23.2 Constants, types and variables

23.2.1 Constants

```
DEFDIALECT = 3
```

Default dialect that will be used when connecting to databases. See TIBConnection.Dialect (663) for more details on dialects.

```
MAXBLOBSEGMENTSIZE = 65535
```

Maximum size to use when fetching blob segments.

23.2.2 Types

```
TStatusVector = Array[0..19] of ISC_STATUS
```

`TStatusVector` is the (opaque) type of the Interbase internal status vector.

23.3 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.

23.4 EIBDatabaseError

23.4.1 Description

Firebird/Interbase database error, a descendant of db.EDatabaseError (658).

See also: db.EDatabaseError (658)

23.4.2 Property overview

Page	Properties	Access	Description
659	GDSErrorCode	r	Firebird/Interbase GDS error code.
659	StatusVector	rw	Low-level IB API status vector.

23.4.3 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.

23.4.4 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.

23.5 TIBConnection

23.5.1 Description

`TIBConnection` is a descendant of TSQLConnection (658) 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 (658)

23.5.2 Method overview

Page	Method	Description
661	Create	Creates a `TIBConnection` object.
662	CreateDB	Creates a database on disk.
662	DropDB	Deletes a database from disk.
661	GetConnectionInfo	Return some information about the connection.

23.5.3 Property overview

Page	Properties	Access	Description
662	BlobSegmentSize	rw	Write this amount of bytes per BLOB segment.
664	CheckTransactionParams	rw	Let StartTransaction check transaction parameters.
663	DatabaseName		Name of the database to connect to.
663	Dialect	rws	Database dialect.
664	KeepConnection		Keep open connection after first query.
664	LoginPrompt		Switch for showing custom login prompt.
663	ODSMajorVersion	r	Database On-Disk Structure major version.
665	OnLogin		Event triggered when a login prompt needs to be shown.
664	Params		Firebird/Interbase specific parameters.
665	Port	s	Port at which the server listens.
665	UseConnectionCharSetIfNone	rw	For string/blob fields with codepage none, use the connection character set when copying data.
665	WireCompression	rw	Use wire compression when communicating with the server.

23.5.4 TIBConnection.Create

Synopsis: Creates a `TIBConnection` object.

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

Visibility: public

Description: Creates a `TIBConnection` object.

23.5.5 TIBConnection.GetConnectionInfo

Synopsis: Return some information about the connection.

Declaration:
```
function GetConnectionInfo(InfoType: TConnInfoType) : string;  Override
```

Visibility: public

Description: `GetConnectionInfo` overrides TSQLConnection.GetConnectionInfo (875) to return the relevant information for the Interbase/Firebird connection.

See also: TSQLConnection.GetConnectionInfo (875), TConnInfoType (854)

23.5.6 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 (658) (specifically, `PAGE_SIZE`) and TSQLConnection.CharSet (658) 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 (658), TSQLConnection.UserName (658), TSQLConnection.Password (658).

If creating a database using Firebird embedded, make sure the embedded library is loaded, the TSQLConnection.HostName (658) property is empty, and set the TSQLConnection.UserName (658) to e.g. 'SYSDBA'. See `TIBConnection`: Firebird/Interbase specific TSQLConnection (658). for details on loading the embedded database library.

See also: TSQLConnection.Params (658), TSQLConnection.DropDB (658), TIBConnection (660)

23.5.7 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 (660).

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 (658), TSQLConnection.UserName (658), TSQLConnection.Password (658).

When using Firebird embedded, make sure the embedded connection library is loaded, the TSQLConnection.HostName (658) property is empty, and set the TSQLConnection.UserName (658) to e.g. 'SYSDBA'. See `TIBConnection`: Firebird/Interbase specific TSQLConnection (658). for more details on loading the embedded library.

See also: TSQLConnection.CreateDB (658), TSQLConnection.HostName (658), TSQLConnection.UserName (658), TSQLConnection.Password (658)

23.5.8 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.

23.5.9 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.

23.5.10 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).

23.5.11 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.

23.5.12 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.

23.5.13 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.

23.5.14 TIBConnection.LoginPrompt

Synopsis: Switch for showing custom login prompt.

Declaration: `Property LoginPrompt :`

Visibility: published

Access:

Description: If true, the OnLogin (658) event will fire, allowing you to handle supplying of credentials yourself.

See also: TSQLConnection.OnLogin (658)

23.5.15 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 (882)

23.5.16 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 (658) property is True, after the BeforeConnect (399) event, but before the connection is actually established.

See also: #fcl.db.TCustomConnection.BeforeConnect (399), TSQLConnection.LoginPrompt (658), #fcl.db.TCustomConnection.Open (396), TSQLConnection.OnLogin (658)

23.5.17 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.

23.5.18 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 (879)

23.5.19 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.

23.6 TIBConnectionDef

23.6.1 Description

Child of TConnectionDef (658) 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 (658)

23.6.2 Method overview

Page	Method	Description
666	ConnectionClass	Firebird/Interbase child of ConnectionClass (861).
667	DefaultLibraryName	Default name of the firebird client library.
666	Description	Description for the Firebird/Interbase child of #fcl.sqldb.TConnectionDef.ConnectionClass (861).
667	LoadedLibraryName	Actally loaded library name.
667	LoadFunction	Return Function to call when loading firebird support.
666	TypeName	Firebird/Interbase child of TConnectionDef.TypeName (658).
667	UnLoadFunction	Return Function to call when unloading firebird support.

23.6.3 TIBConnectionDef.TypeName

Synopsis: Firebird/Interbase child of TConnectionDef.TypeName (658).

Declaration: `class function TypeName : string;  Override`

Visibility: default

See also: TConnectionDef.TypeName (658), TIBConnection (660)

23.6.4 TIBConnectionDef.ConnectionClass

Synopsis: Firebird/Interbase child of ConnectionClass (861).

Declaration: `class function ConnectionClass : TSQLConnectionClass;  Override`

Visibility: default

See also: TConnectionDef.ConnectionClass (658), TIBConnection (660)

23.6.5 TIBConnectionDef.Description

Synopsis: Description for the Firebird/Interbase child of #fcl.sqldb.TConnectionDef.ConnectionClass (861).

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 (861), TIBConnection (660)

23.6.6 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.

23.6.7 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.

23.6.8 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.

23.6.9 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 (667)

23.7 TIBCursor

23.7.1 Description

A cursor that keeps track of where you are in a Firebird/Interbase dataset. It is a descendent of TSQLCursor (658).

See also: TSQLCursor (658), TIBConnection (660)

23.8 TIBTrans

23.8.1 Description

Firebird/Interbase database transaction object. Descendant of TSQLHandle (658).

See also: TSQLHandle (658), TIBConnection (660)

Chapter 24

Reference for unit 'idea'

24.1 Used units

Table 24.1: Used units by unit 'idea'

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

24.2 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.

24.3 Constants, types and variables

24.3.1 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.

24.3.2 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 (671) or DeKeyIDEA (670) 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.

24.4 Procedures and functions

24.4.1 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 (671), DeKeyIdea (670), TIDEAEncryptStream (673), TIDEADecryptStream (671)

24.4.2 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 (671), CipherIdea (670)

24.4.3 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 (670), CipherIdea (670)

24.5 EIDEAError

24.5.1 Description

`EIDEAError` is used to signal errors in the IDEA encryption decryption streams.

24.6 TIDEADeCryptStream

24.6.1 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 (670) 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 (673) encryption stream.

See also: TIDEAEncryptStream (673), TIDEAStream.Create (675), CipherIDEA (670)

24.6.2 Method overview

Page	Method	Description
672	Create	Constructor to create a new `TIDEADecryptStream` instance.
672	Read	Reads data from the stream, decrypting it as needed.
672	Seek	Set position on the stream.

24.6.3 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 (670), which is then passed on to the inherited constructor TIDEAStream.Create (675). It is an easy-access function which introduces no new functionality.

The string is truncated at the maximum length of the TIdeaCryptKey (670) structure, so it makes no sense to provide a string with length longer than this structure.

See also: TIdeaCryptKey (670), TIDEAStream.Create (675), TIDEAEnCryptStream.Create (673)

24.6.4 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 (672), TIDEAEncryptStream (673)

24.6.5 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 (671) exception is raised if the stream does not allow the requested seek operation.

See also: Read (672)

24.7 TIDEAEncryptStream

24.7.1 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 (670) 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 (671) decryption stream.

See also: TIDEADecryptStream (671), TIDEAStream.Create (675), CipherIDEA (670)

24.7.2 Method overview

Page	Method	Description
673	Create	Constructor to create a new `TIDEAEncryptStream` instance.
673	Destroy	Flush data buffers and free the stream instance.
674	Flush	Write remaining bytes from the stream.
674	Seek	Set stream position.
674	Write	Write bytes to the stream to be encrypted.

24.7.3 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 (670), which is then passed on to the inherited constructor TIDEAStream.Create (675). It is an easy-access function which introduces no new functionality.

The string is truncated at the maximum length of the TIdeaCryptKey (670) structure, so it makes no sense to provide a string with length longer than this structure.

See also: TIdeaCryptKey (670), TIDEAStream.Create (675), TIDEADeCryptStream.Create (672)

24.7.4 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 (675)

24.7.5 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 (672)

24.7.6 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 (671) will be raised unless called with `0` and `soFromCurrent` as arguments.

See also: Write (674), EIDEAError (671)

24.7.7 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 (674)

24.8 TIDEAStream

24.8.1 Description

Do not create instances of `TIDEAStream` directly. It implements no useful functionality: it serves as a common ancestor of the TIDEAEncryptStream (673) and TIDEADeCryptStream (671), 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 (673), TIDEADeCryptStream (671)

24.8.2 Method overview

Page	Method	Description
675	Create	Creates a new instance of the `TIDEAStream` class.

24.8.3 Property overview

Page	Properties	Access	Description
675	Key	r	Key used when encrypting/decrypting.

24.8.4 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 (673), TIDEADeCryptStream (671)

24.8.5 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 (670)

Chapter 25

Reference for unit 'inicol'

25.1 Used units

Table 25.1: Used units by unit 'inicol'

Name	Page
Classes	??
IniFiles	686
System	??
sysutils	??

25.2 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 (688) class for this.

25.3 Constants, types and variables

25.3.1 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.

25.4 EIniCol

25.4.1 Description

`EIniCol` is used to report error conditions in the load and save methods of TIniCollection (677).

25.5 TIniCollection

25.5.1 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 (681).

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 (679), TIniCollection.LoadFromIni (679), TIniCollection.SaveToIni (678), TIniCollection.SaveToFile (678)

25.5.2 Method overview

Page	Method	Description
677	Load	Loads the collection from the default filename.
679	LoadFromFile	Load collection from file.
679	LoadFromIni	Load collection from a file in .ini file format.
678	Save	Save the collection to the default filename.
678	SaveToFile	Save collection to a file in .ini file format.
678	SaveToIni	Save the collection to a `TCustomIniFile` descendent.

25.5.3 Property overview

Page	Properties	Access	Description
680	FileName	rw	Filename of the collection.
680	GlobalSection	rw	Name of the global section.
679	Prefix	r	Prefix used in global section.
680	SectionPrefix	r	Prefix string for section names.

25.5.4 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 (680) property. It calls the LoadFromFile (679) 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 (679), TIniCollection.LoadFromIni (679), TIniCollection.Save (678), FileName (680)

25.5.5 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 (680) property, using GlobalSection (680) as the section. It calls the SaveToFile (678) 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 (678), TIniCollection.SaveToIni (678), TIniCollection.Load (677), FileName (680)

25.5.6 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 (679) property for the key. After this it calls the SaveToIni (681) method of all TIniCollectionItem (681) 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 (680) property contains the value of `Section`, it will be used in the Save (678) method.

See also: TIniCollectionItem.SaveToIni (681), TIniCollection.Save (678)

25.5.7 TIniCollection.SaveToFile

Synopsis: Save collection to a file in .ini file format.

Declaration: `procedure SaveToFile(AFileName: string; 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 (678) 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 (678), TIniCollection.LoadFromFile (679)

25.5.8 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 (681) 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 (678) method description.

See also: TIniCollection.LoadFromFile (679), TIniCollectionItem.LoadFromIni (681), TIniCollection.SaveToIni (678)

25.5.9 TIniCollection.LoadFromFile

Synopsis: Load collection from file.

Declaration: `procedure LoadFromFile(AFileName: string; Section: string)`

Visibility: public

Description: `LoadFromFile` creates a `TMemIniFile` instance using `AFileName` as the filename. It calls LoadFromIni (679) using this instance and `Section` as the parameters.

See also: TIniCollection.LoadFromIni (679), TIniCollection.Load (677), TIniCollection.SaveToIni (678), TIniCollection.SaveToFile (678)

25.5.10 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 (680), TIniCollection.GlobalSection (680)

25.5.11 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 (682) 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 (680), TIniCollectionItem.SectionName (682)

25.5.12 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 (679) or SaveToFile (678) operation. It is used in the Load (677) or Save (678) calls.

See also: Save (678), LoadFromFile (679), SaveToFile (678), Load (677)

25.5.13 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 (679) or SaveToIni (678) calls. It's used in the Load (677) or Save (678) calls.

See also: Save (678), LoadFromFile (679), SaveToFile (678), Load (677)

25.6 TIniCollectionItem

25.6.1 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 (681) and LoadFromIni (681) methods should be overridden. They should implement the actual loading and saving. The loading and saving is always initiated by the methods in TIniCollection (677), TIniCollection.LoadFromIni (679) and TIniCollection.SaveToIni (678) respectively.

See also: TIniCollection (677), TIniCollectionItem.SaveToIni (681), TIniCollectionItem.LoadFromIni (681), TIniCollection.LoadFromIni (679), TIniCollection.SaveToIni (678)

25.6.2 Method overview

Page	Method	Description
682	LoadFromFile	Load item from a file.
681	LoadFromIni	Method called when the item must be loaded.
682	SaveToFile	Save item to a file.
681	SaveToIni	Method called when the item must be saved.

25.6.3 Property overview

Page	Properties	Access	Description
682	SectionName	rw	Default section name.

25.6.4 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 (678) 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 (678), LoadFromIni (681), SaveToFile (682), LoadFromFile (682)

25.6.5 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 (679) 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 (679), SaveToIni (681), LoadFromFile (682), SaveToFile (682)

25.6.6 TIniCollectionItem.SaveToFile

Synopsis: Save item to a file.

Declaration: `procedure SaveToFile(FileName: string; Section: string)`

Visibility: public

Description: `SaveToFile` creates an instance of `TIniFile` with the indicated `FileName` calls SaveToIni (681) 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 (681), LoadFromFile (682)

25.6.7 TIniCollectionItem.LoadFromFile

Synopsis: Load item from a file.

Declaration: `procedure LoadFromFile(FileName: string; Section: string)`

Visibility: public

Description: `LoadFromFile` creates an instance of `TMemIniFile` and calls LoadFromIni (681) to load the item from the indicated file in .ini format from the section `Section`.

Errors: None.

See also: SaveToFile (682), LoadFromIni (681)

25.6.8 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 (682), LoadFromIni (681)

25.7 TNamedIniCollection

25.7.1 Description

`TNamedIniCollection` is the collection to go with the TNamedIniCollectionItem (684) item class. it provides some functions to look for items based on the UserData (683) or based on the Name (683).

See also: TNamedIniCollectionItem (684), IndexOfUserData (683), IndexOfName (683)

25.7.2 Method overview

Page	Method	Description
684	FindByName	Return the item based on its name.
684	FindByUserData	Return the item based on its `UserData`.
683	IndexOfName	Search for an item, based on its name, and return its position.
683	IndexOfUserData	Search for an item based on it's `UserData` property.

25.7.3 Property overview

Page	Properties	Access	Description
684	NamedItems	rw	Indexed access to the `TNamedIniCollectionItem` items.

25.7.4 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 (683) 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 (683), TNamedIniCollectionItem.UserData (685)

25.7.5 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 (683), TNamedIniCollectionItem.Name (685)

25.7.6 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 (683) 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 (683), FindByUserData (684)

25.7.7 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 (685) property value matches the `UserData` parameter. If no item is found, `Nil` is returned.

Errors: If no item is found, `Nil` is returned.

25.7.8 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 (684) items. The index is zero based.

See also: TNamedIniCollectionItem (684)

25.8 TNamedIniCollectionItem

25.8.1 Description

`TNamedIniCollectionItem` is a TIniCollectionItem (681) 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 (681)

25.8.2 Property overview

Page	Properties	Access	Description
685	Name	rw	Name of the item.
685	UserData	rw	User-defined data.

25.8.3 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`.

25.8.4 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 (685)

Chapter 26

Reference for unit 'IniFiles'

26.1 Used units

Table 26.1: Used units by unit 'IniFiles'

Name	Page
Classes	??
Contnrs	213
System	??
sysutils	??

26.2 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 (688) and is implemented in TIniFile (701) and TMemIniFile (710). The API presented by these components is Delphi compatible.

26.3 Constants, types and variables

26.3.1 Types

```
TIniFileOption = (ifoStripComments,ifoStripInvalid,ifoEscapeLineFeeds
  ,
                  ifoCaseSensitive,ifoStripQuotes,
  ifoFormatSettingsActive,ifoWriteStringBoolean)
```

Table 26.2: 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.
ifoStripComments	Strip comments from file.
ifoStripInvalid	Strip invalid lines from file.
ifoStripQuotes	Strip double quotes from values.
ifoWriteStringBoolean	Read/Write booleans as strings instead of 0/1.

`TIniFileOption` enumerates the possible options when creating a new TCustomIniFile (688) 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 (686). It is used in the TCUstomIniFile.Create (689) constructor and TCustomIniFile.Options (698) property.

```
TSectionValuesOption = (svoIncludeComments,svoIncludeInvalid,
  svoIncludeQuotes)
```

Table 26.3: 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 (696)

svoIncludeComments Include comment lines.

svoIncludeInvalid Include invalid lines.

svoIncludeQuotes Include existing quotes around values.

```
TSectionValuesOptions = Set of TSectionValuesOption
```

`TSectionValuesOptions` is the set for TSectionValuesOptions (687). It is used in the TCUstomIniFile.ReadSectionValues (696) call.

26.4 TCustomIniFile

26.4.1 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 (701) or TMemIniFile (710).

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 (701), TMemIniFile (710)

26.4.2 Method overview

Page	Method	Description
689	Create	Instantiate a new instance of `TCustomIniFile`.
697	DeleteKey	Delete a key from a section.
690	Destroy	Remove the `TCustomIniFile` instance from memory.
696	EraseSection	Clear a section.
694	ReadBinaryStream	Read binary data.
692	ReadBool	
693	ReadDate	Read a date value.
693	ReadDateTime	Read a Date/Time value.
693	ReadFloat	Read a floating point value.
691	ReadInt64	Read an Int64 value.
691	ReadInteger	Read an integer value from the file.
695	ReadSection	Read the key names in a section.
696	ReadSections	Read the list of sections.
696	ReadSectionValues	Read names and values of a section.
690	ReadString	Read a string valued key.
693	ReadTime	Read a time value.
690	SectionExists	Check if a section exists.
690	SetBoolStringValues	Set the boolean string values to use when writing to file.
697	UpdateFile	Update the file on disk.
697	ValueExists	Check if a value exists.
695	WriteBinaryStream	Write binary data.
692	WriteBool	Write boolean value.
694	WriteDate	Write date value.
694	WriteDateTime	Write date/time value.
695	WriteFloat	Write a floating-point value.
692	WriteInt64	Write an Int64 value.
691	WriteInteger	Write an integer value.
691	WriteString	Write a string value.
695	WriteTime	Write time value.

26.4.3 Property overview

Page	Properties	Access	Description
700	BoolFalseStrings	rw	Strings to recognize as boolean `False` values.
699	BoolTrueStrings	rw	Strings to recognize as boolean `True` values.
698	CaseSensitive	rw	Are key and section names case sensitive.
697	Encoding	rw	Encoding of the ini file.
698	EscapeLineFeeds	r	Should linefeeds be escaped ?
698	FileName	r	Name of the .ini file.
699	FormatSettingsActive	rw	Is `FormatSettings` used or not.
698	Options	rw	Options currently in effect.
700	OwnsEncoding	r	Does the ini file instance own the encoding ?
699	StripQuotes	rw	Should quotes be stripped from string values.

26.4.4 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 (698) 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 (690), TCustomIniFile.Options (698)

26.4.5 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 (688)

26.4.6 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 (692). 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 (698).

See also: TCustomIniFile.ReadBool (692), TCustomIniFile.WriteBool (692), TCustomIniFile.BoolTrueStrings (699), TCustomIniFile.BoolFalseStrings (700), TCustomIniFile.Options (698)

26.4.7 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 (697)

26.4.8 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 (691), ReadInteger (691), ReadBool (692), ReadDate (693), ReadDateTime (693), ReadTime (693), ReadFloat (693), ReadBinaryStream (694)

26.4.9 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 (690), WriteInteger (691), WriteBool (692), WriteDate (694), WriteDateTime (694), WriteTime (695), WriteFloat (695), WriteBinaryStream (695)

26.4.10 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 (691), ReadString (690), ReadBool (692), ReadDate (693), ReadDateTime (693), ReadTime (693), ReadFloat (693), ReadBinaryStream (694)

26.4.11 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 (691), WriteString (691), WriteBool (692), WriteDate (694), WriteDateTime (694), WriteTime (695), WriteFloat (695), WriteBinaryStream (695)

26.4.12 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 (691) reads at most a 32-bit value.

See also: TCustomIniFile.ReadInteger (691), TCustomIniFile.WriteInt64 (692)

26.4.13 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 (691), TCustomIniFile.ReadInt64 (691)

26.4.14 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 (692), ReadInteger (691), ReadString (690), ReadDate (693), ReadDateTime (693), ReadTime (693), ReadFloat (693), ReadBinaryStream (694)

26.4.15 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 (692), WriteInteger (691), WriteString (691), WriteDate (694), WriteDateTime (694), WriteTime (695), WriteFloat (695), WriteBinaryStream (695)

26.4.16 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 (694), ReadInteger (691), ReadBool (692), ReadString (690), ReadDateTime (693), ReadTime (693), ReadFloat (693), ReadBinaryStream (694)

26.4.17 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 (694), ReadInteger (691), ReadBool (692), ReadDate (693), ReadString (690), ReadTime (693), ReadFloat (693), ReadBinaryStream (694)

26.4.18 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 (695), ReadInteger (691), ReadBool (692), ReadDate (693), ReadDateTime (693), ReadTime (693), ReadString (690), ReadBinaryStream (694)

26.4.19 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 (695), ReadInteger (691), ReadBool (692), ReadDate (693), ReadDateTime (693), ReadString (690), ReadFloat (693), ReadBinaryStream (694)

26.4.20 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 (695), ReadInteger (691), ReadBool (692), ReadDate (693), ReadDateTime (693), ReadTime (693), ReadFloat (693), ReadString (690)

26.4.21 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 (693), WriteInteger (691), WriteBool (692), WriteString (691), WriteDateTime (694), WriteTime (695), WriteFloat (695), WriteBinaryStream (695)

26.4.22 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 (693), WriteInteger (691), WriteBool (692), WriteDate (694), WriteString (691), WriteTime (695), WriteFloat (695), WriteBinaryStream (695)

26.4.23 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 (693), WriteInteger (691), WriteBool (692), WriteDate (694), WriteDateTime (694), WriteTime (695), WriteString (691), WriteBinaryStream (695)

26.4.24 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 (693), WriteInteger (691), WriteBool (692), WriteDate (694), WriteDateTime (694), WriteString (691), WriteFloat (695), WriteBinaryStream (695)

26.4.25 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 (694), WriteInteger (691), WriteBool (692), WriteDate (694), WriteDateTime (694), WriteTime (695), WriteFloat (695), WriteString (691)

26.4.26 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 (696) method should be used.

See also: ReadSections (696), SectionExists (690), ReadSectionValues (696)

26.4.27 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 (690), ReadSectionValues (696), ReadSection (695)

26.4.28 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 (695) can be used.

See also: SectionExists (690), ReadSections (696), ReadSection (695)

26.4.29 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 (690), ReadSections (696), DeleteKey (697)

26.4.30 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 (696)

26.4.31 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 (705)

26.4.32 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 (690)

26.4.33 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 (689), TCustomIniFile.OwnsEncoding (700)

26.4.34 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 (689)

26.4.35 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 (686) for a list of allowed options. The initial value of this property can be specified using the constructor of the class, TCustomIniFile.Create (689). Not all options can be specified after the ini file object was created.

See also: TInifileOption (686), TInifileOptions (687), TCustomIniFile.Create (689)

26.4.36 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 (689), as the value of this property must be specified when the `TCustomIniFile` instance is created.

By default, `EscapeLineFeeds` is `False`.

See also: Create (689), CaseSensitive (698)

26.4.37 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 (698)

26.4.38 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).

26.4.39 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.

26.4.40 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 (692) The first string in the list will be used when writing a `True` boolean value in TCustomIniFile.WriteBool (692).

The string values are ignored if `ifoWriteStringBoolean` is not in the ini file options (698).

See also: TCustomIniFile.SetBoolStringValues (690), TCustomIniFile.BoolFalseStrings (700), TCustomIniFile.ReadBool (692), TCustomIniFile.WriteBool (692), TCustomIniFile.Options (698)

26.4.41 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 (692) The first string in the list will be used when writing a `False` boolean value in TCustomIniFile.WriteBool (692).

The string values are ignored if `ifoWriteStringBoolean` is not in the ini file options (698).

See also: TCustomIniFile.SetBoolStringValues (690), TCustomIniFile.BoolTrueStrings (699), TCustomIniFile.ReadBool (692), TCustomIniFile.WriteBool (692), TCustomIniFile.Options (698)

26.4.42 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 (697), TCustomIniFile.Create (689)

26.5 THashedStringList

26.5.1 Description

`THashedStringList` is a TStringList (**??**) descendent which creates has values for the strings and names (in the case of a name-value pair) stored in it. The IndexOf (701) and IndexOfName (701) functions make use of these hash values to quickly locate a value.

See also: IndexOf (701), IndexOfName (701)

26.5.2 Method overview

Page	Method	Description
700	Destroy	Clean up instance.
701	IndexOf	Returns the index of a string in the list of strings.
701	IndexOfName	Return the index of a name in the list of name=value pairs.

26.5.3 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`.

26.5.4 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 (701)

26.5.5 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 (701)

26.6 TIniFile

26.6.1 Description

`TIniFile` is an implementation of TCustomIniFile (688) which does the same as TMemIniFile (710), 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 (688), TMemIniFile (710)

26.6.2 Method overview

Page	Method	Description
702	Create	Create a new instance of `TIniFile`.
704	DeleteKey	Delete key.
702	Destroy	Remove the `TIniFile` instance from memory.
704	EraseSection	
703	ReadSection	Read the key names in a section.
703	ReadSectionRaw	Read raw section.
703	ReadSections	Read section names.
704	ReadSectionValues	
702	ReadString	Read a string.
704	UpdateFile	Update the file on disk.
703	WriteString	Write string to file.

26.6.3 Property overview

Page	Properties	Access	Description
705	CacheUpdates	rw	Should changes be kept in memory.
705	Stream	r	Stream from which ini file was read.
705	WriteBOM	rw	Indicates if a Byte Order Mark (BOM) is written at the start of the .INI file.

26.6.4 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 (698) property can be specified in `AOptions`.

See also: TCustomIniFile (688), TMemIniFile (710), TCustomIniFile.Options (698)

26.6.5 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 (697), CacheUpdates (705)

26.6.6 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 (690) abstract method by looking at the in-memory copy of the ini file and returning the string found there.

See also: TCustomIniFile.ReadString (690)

26.6.7 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 (691) abstract method by writing the string to the in-memory copy of the ini file. If CacheUpdates (705) 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.

26.6.8 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 (695)

See also: TCustomIniFile.ReadSection (695), TIniFile.ReadSectionRaw (703)

26.6.9 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 (703), TCustomIniFile.ReadSection (695)

26.6.10 TIniFile.ReadSections

Synopsis: Read section names.

Declaration:
```
procedure ReadSections(Strings: TStrings);  Override
```

Visibility: public

Description: `ReadSections` is the implementation of TCustomIniFile.ReadSections (696). It operates on the in-memory copy of the inifile, and places all section names in `Strings`.

See also: TIniFile.ReadSection (703), TCustomIniFile.ReadSections (696), TIniFile.ReadSectionValues (704)

26.6.11 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 (696). 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 (703), TCustomIniFile.ReadSectionValues (696), TIniFile.ReadSections (703)

26.6.12 TIniFile.EraseSection

Synopsis:

Declaration:
```
procedure EraseSection(const Section: string);  Override
```

Visibility: public

Description: `Erasesection` deletes the section `Section` from memory, if CacheUpdates (705) is `False`, then the file is immediately updated on disk. This method is the implementation of the abstract TCustomIniFile.EraseSection (696) method.

See also: TCustomIniFile.EraseSection (696), TIniFile.ReadSection (703), TIniFile.ReadSections (703)

26.6.13 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 (705) is `False`, then the file is immediately updated on disk.

See also: CacheUpdates (705)

26.6.14 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 (705)

26.6.15 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 (704) method will use this stream to write changes to.

See also: Create (702), UpdateFile (704)

26.6.16 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 (704). 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 (704)

26.6.17 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.

26.7 TIniFileKey

26.7.1 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 (701) class.

See also: TIniFile (701)

26.7.2 Method overview

Page	Method	Description
706	Create	Create a new instance of `TIniFileKey`.

26.7.3 Property overview

Page	Properties	Access	Description
706	Ident	rw	Key name.
706	Value	rw	Key value.

26.7.4 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 (706) with `AIdent` and Value (706) with `AValue`.

See also: Ident (706), Value (706)

26.7.5 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 (706)

26.7.6 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 (706)

26.8 TIniFileKeyList

26.8.1 Description

`TIniFileKeyList` maintains a list of TIniFileKey (706) instances on behalf of the TIniFileSection (708) class. It stores they keys of one section of the .ini files.

See also: TIniFileKey (706), TIniFileSection (708)

26.8.2 Method overview

Page	Method	Description
707	Clear	Clear the list.
707	Destroy	Free the instance.

26.8.3 Property overview

Page	Properties	Access	Description
707	Items	r	Indexed access to `TIniFileKey` items in the list.

26.8.4 TIniFileKeyList.Destroy

Synopsis: Free the instance.

Declaration: `destructor Destroy;  Override`

Visibility: public

Description: `Destroy` clears up the list using Clear (707) and then calls the inherited destroy.

See also: Clear (707)

26.8.5 TIniFileKeyList.Clear

Synopsis: Clear the list.

Declaration: `procedure Clear;  Override`

Visibility: public

Description: `Clear` removes all TIniFileKey (706) instances from the list, and frees the instances.

See also: TIniFileKey (706)

26.8.6 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 (706) items in the list. The index is zero-based and runs from 0 to `Count-1`.

See also: TIniFileKey (706)

26.9 TIniFileSection

26.9.1 Description

`TIniFileSection` is a class which represents a section in the .ini, and is used internally by the TIniFile (701) class (one instance of `TIniFileSection` is created for each section in the file by the TIniFileSectionList (709) list). The name of the section is stored in the Name (709) property, and the key/value pairs in this section are available in the KeyList (709) property.

See also: TIniFileKeyList (707), TIniFile (701), TIniFileSectionList (709)

26.9.2 Method overview

Page	Method	Description
708	Create	Create a new section object.
709	Destroy	Free the section object from memory.
708	Empty	Is the section empty.

26.9.3 Property overview

Page	Properties	Access	Description
709	KeyList	r	List of key/value pairs in this section.
709	Name	r	Name of the section.

26.9.4 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.

26.9.5 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 (707) instance to keep all the key/value pairs for this section.

See also: TIniFileKeyList (707)

26.9.6 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 (708), TIniFileKeyList (707)

26.9.7 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 (709)

26.9.8 TIniFileSection.KeyList

Synopsis: List of key/value pairs in this section.

Declaration: `Property KeyList : TIniFileKeyList`

Visibility: public

Access: Read

Description: `KeyList` is the TIniFileKeyList (707) instance that is used by the `TIniFileSection` to keep the key/value pairs of the section.

See also: TIniFileSection.Name (709), TIniFileKeyList (707)

26.10 TIniFileSectionList

26.10.1 Description

`TIniFileSectionList` maintains a list of TIniFileSection (708) instances, one for each section in an .ini file. `TIniFileSectionList` is used internally by the TIniFile (701) class to represent the sections in the file.

See also: TIniFileSection (708), TIniFile (701)

26.10.2 Method overview

Page	Method	Description
710	Clear	Clear the list.
710	Destroy	Free the object from memory.

26.10.3 Property overview

Page	Properties	Access	Description
710	Items	r	Indexed access to all the section objects in the list.

26.10.4 TIniFileSectionList.Destroy

Synopsis: Free the object from memory.

Declaration: `destructor Destroy;  Override`

Visibility: public

Description: `Destroy` calls Clear (710) to clear the section list and the calls the inherited `Destroy`

See also: Clear (710)

26.10.5 TIniFileSectionList.Clear

Synopsis: Clear the list.

Declaration: `procedure Clear;  Override`

Visibility: public

Description: `Clear` removes all TIniFileSection (708) items from the list, and frees the items it removes from the list.

See also: TIniFileSection (708), TIniFileSectionList.Items (710)

26.10.6 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 (708), TIniFileSectionList.Clear (710)

26.11 TMemIniFile

26.11.1 Description

`TMemIniFile` is a simple descendent of TIniFile (701) 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 (701), TCustomIniFile (688), CacheUpdates (705)

26.11.2 Method overview

Page	Method	Description
711	Clear	Clear the data.
711	Create	Create a new instance of `TMemIniFile`.
711	GetStrings	Get contents of ini file as stringlist.
712	Rename	Rename the ini file.
712	SetStrings	Set data from a stringlist.

26.11.3 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 (702), and sets the CacheUpdates (705) to `True` so updates will be kept in memory till they are explicitly written to disk.

See also: TIniFile.Create (702), CacheUpdates (705)

26.11.4 TMemIniFile.Clear

Synopsis: Clear the data.

Declaration:
```
procedure Clear
```

Visibility: public

Description: `Clear` removes all sections and key/value pairs from memory. If CacheUpdates (705) is set to `False` then the file on disk will immediately be emptied.

See also: SetStrings (712), GetStrings (711)

26.11.5 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 (712), which can be used to copy data between 2 in-memory ini files.

See also: SetStrings (712), Clear (711)

26.11.6 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 (704)

26.11.7 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 (711). The two calls combined can be used to copy data between 2 in-memory ini files.

See also: GetStrings (711), Clear (711)

26.12 TStringHash

26.12.1 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 (701), TStringHash.Create (713), TStringHash.ValueOf (714)

26.12.2 Method overview

Page	Method	Description
713	Add	Add a new value to the hash.
713	Clear	Remove all values.
713	Create	Create a new instance of `TStringHash`.
713	Destroy	Free `TStringHash` instance.
714	Modify	Try to modify an existing value.
714	Remove	Remove a key from the hash.
714	ValueOf	Retrieve value of `Key`.

26.12.3 Property overview

Page	Properties	Access	Description
714	AddReplacesExisting	rw	Should `Add` replace existing values or not.

26.12.4 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 (714), TStringHash.Destroy (713)

26.12.5 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 (713), TStringHash.Clear (713)

26.12.6 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 (714). 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 (714), TStringHash.Modify (714), TStringHash.Remove (714), TStringHash.ValueOf (714)

26.12.7 TStringHash.Clear

Synopsis: Remove all values.

Declaration: `procedure Clear`

Visibility: public

Description: `Clear` removes all values from the hash.

See also: TStringHash.Destroy (713), TStringHash.Add (713), TStringHash.Remove (714), TStringHash.ValueOf (714)

26.12.8 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 (713), TStringHash.Clear (713), TStringHash.Remove (714), TStringHash.ValueOf (714)

26.12.9 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 (713), TStringHash.Clear (713), TStringHash.Modify (714), TStringHash.ValueOf (714)

26.12.10 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 (713), TStringHash.Clear (713), TStringHash.Modify (714), TStringHash.Remove (714)

26.12.11 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 (713) will replace an existing value (`True`) or will raise an exception when an existing value is added (`False`).

See also: TStringHash.Add (713)

Chapter 27

Reference for unit 'iostream'

27.1 Used units

Table 27.1: Used units by unit 'iostream'

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

27.2 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`).

27.3 Constants, types and variables

27.3.1 Types

```
TIOSType = (iosInput,iosOutPut,iosError)
```

Table 27.2: 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 (716) constructor of TIOStream (716), it determines what kind of stream is created.

27.4 EIOStreamError

27.4.1 Description

Error thrown in case of an invalid operation on a TIOStream (716).

27.5 TIOStream

27.5.1 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 (715) 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 (717) behaviour based on this position.

See also: TIOSType (715), THandleStream (**??**)

27.5.2 Method overview

Page	Method	Description
716	Create	Construct a new instance of TIOStream (716).
716	Read	Read data from the stream.
717	Seek	Set the stream position.
717	Write	Write data to the stream.

27.5.3 TIOStream.Create

Synopsis: Construct a new instance of TIOStream (716).

Declaration: `constructor Create(aIOSType: TIOSType)`

Visibility: public

Description: `Create` creates a new instance of TIOStream (716), 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 (716), TIOStream.Write (717)

27.5.4 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 (716) 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 (715), TIOStream.Write (717)

27.5.5 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 (716) 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 (715), TIOStream.Read (716)

27.5.6 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 (716) exception is raised if the stream does not allow the requested seek operation.

See also: EIOStreamError (716)

Chapter 28

Reference for unit 'libtar'

28.1 Used units

Table 28.1: Used units by unit 'libtar'

Name	Page
BaseUnix	??
Classes	??
System	??
sysutils	??
Unix	??
UnixType	??

28.2 Overview

The libtar units provides 2 classes to read and write .tar archives: TTarArchive (722) class can be used to read a tar file, and the TTarWriter (724) class can be used to write a tar file. The unit was implemented originally by Stefan Heymann.

28.3 Constants, types and variables

28.3.1 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.

28.3.2 Types

```
TFileType = (ftNormal,ftLink,ftSymbolicLink,ftCharacter,ftBlock,
  ftDirectory,ftFifo,ftContiguous,ftDumpDir,ftMultiVolume
  ,
             ftVolumeHeader)
```

Table 28.2: 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 (720) 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 (723) call.

```
TTarMode = (tmSetUid,tmSetGid,tmSaveText)
```

Table 28.3: 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 (720) record.

```
TTarModes = Set of TTarMode
```

`TTarModes` denotes the full set of permission bits for the file in the field `Mode` field of the TTarDirRec (720) record.

```
TTarPermission = (tpReadByOwner,tpWriteByOwner,tpExecuteByOwner,
  tpReadByGroup,tpWriteByGroup,tpExecuteByGroup,
  tpReadByOther,tpWriteByOther,tpExecuteByOther)
```

Table 28.4: 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 (720) record.

28.4 Procedures and functions

28.4.1 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 (720)

28.4.2 ConvertFilename

Synopsis: Convert filename to archive format.

Declaration:
```
function ConvertFilename(Filename: string) : string
```

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.

28.4.3 FileTimeGMT

Synopsis: Extract filetime.

Declaration:
```
function FileTimeGMT(FileName: string) : 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 (720)

28.4.4 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 (721)

28.5 TTarArchive

28.5.1 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 (724), FindNext (723)

28.5.2 Method overview

Page	Method	Description
722	Create	Create a new instance of the archive.
722	Destroy	Destroy `TTarArchive` instance.
723	FindNext	Find next archive entry.
723	GetFilePos	Return current archive position.
723	ReadFile	Read a file from the archive.
722	Reset	Reset archive.
724	SetFilePos	Set position in archive.

28.5.3 TTarArchive.Create

Synopsis: Create a new instance of the archive.

Declaration:
```
constructor Create(Stream: TStream);  Overload
constructor Create(Filename: string; 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 (723)

28.5.4 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 (722)

28.5.5 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 (722)

28.5.6 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 (723)

28.5.7 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.

28.5.8 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 (724), TTarArchive.Reset (722)

28.5.9 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 (722), TTarArchive.GetFilePos (723)

28.6 TTarWriter

28.6.1 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 (722)

28.6.2 Method overview

Page	Method	Description
726	AddDir	Add directory to archive.
725	AddFile	Add a file to the archive.
727	AddLink	Add hard link to archive.
725	AddStream	Add stream contents to archive.
726	AddString	Add string as file data.
726	AddSymbolicLink	Add a symbolic link to the archive.
727	AddVolumeHeader	Add volume header entry.
724	Create	Create a new archive.
725	Destroy	Close archive and clean up `TTarWriter`.
727	Finalize	Finalize the archive.

28.6.3 Property overview

Page	Properties	Access	Description
728	GID	rw	Archive entry group ID.
728	GroupName	rw	Archive entry group name.
729	Magic	rw	Archive entry Magic constant.
729	Mode	rw	Archive entry mode.
727	Permissions	rw	Archive entry permissions.
728	UID	rw	Archive entry user ID.
728	UserName	rw	Archive entry user name.

28.6.4 TTarWriter.Create

Synopsis: Create a new archive.

Declaration:
```
constructor Create(TargetStream: TStream);  Overload
constructor Create(TargetFilename: string; 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 (725)

28.6.5 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 (727)

28.6.6 TTarWriter.AddFile

Synopsis: Add a file to the archive.

Declaration:
```
procedure AddFile(Filename: string; 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 (725), TTarWriter.AddString (726), TTarWriter.AddLink (727), TTarWriter.AddSymbolicLink (726), TTarWriter.AddDir (726), TTarWriter.AddVolumeHeader (727)

28.6.7 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 (725), TTarWriter.AddString (726), TTarWriter.AddLink (727), TTarWriter.AddSymbolicLink (726), TTarWriter.AddDir (726), TTarWriter.AddVolumeHeader (727)

28.6.8 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 (725), TTarWriter.AddStream (725), TTarWriter.AddLink (727), TTarWriter.AddSymbolicLink (726), TTarWriter.AddDir (726), TTarWriter.AddVolumeHeader (727)

28.6.9 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 (725), TTarWriter.AddStream (725), TTarWriter.AddLink (727), TTarWriter.AddSymbolicLink (726), TTarWriter.AddString (726), TTarWriter.AddVolumeHeader (727)

28.6.10 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 (725), TTarWriter.AddStream (725), TTarWriter.AddLink (727), TTarWriter.AddDir (726), TTarWriter.AddString (726), TTarWriter.AddVolumeHeader (727)

28.6.11 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 (725), TTarWriter.AddStream (725), TTarWriter.AddSymbolicLink (726), TTarWriter.AddDir (726), TTarWriter.AddString (726), TTarWriter.AddVolumeHeader (727)

28.6.12 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 (725), TTarWriter.AddStream (725), TTarWriter.AddSymbolicLink (726), TTarWriter.AddDir (726), TTarWriter.AddString (726), TTarWriter.AddLink (727)

28.6.13 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 (725)

28.6.14 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 (720)

28.6.15 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 (720)

28.6.16 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 (720)

28.6.17 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 (720)

28.6.18 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 (720)

28.6.19 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 (720)

28.6.20 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 (720)

Chapter 29

Reference for unit 'MaskUtils'

29.1 Used units

Table 29.1: Used units by unit 'MaskUtils'

Name	Page
System	??
sysutils	??

29.2 Overview

This unit provides routines to work with edit masks. For this, it provied FormatMaskText (733), FormatMaskInput (732) and MaskDoFormatText (733) functions, as well as the underlying TMaskUtils (733) class.

29.3 Constants, types and variables

29.3.1 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 29.2: 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 (733) 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 29.3: 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 (733) 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.

29.4 Procedures and functions

29.4.1 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 (733), MaskDoFormatText (733), TMaskUtils (733), TMaskUtils.InputMask (735)

29.4.2 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 (733) instance to do the actual work.

Errors: None.

See also: FormatMaskInput (732), MaskDoFormatText (733), TMaskUtils (733), TMaskUtils.ApplyMaskToText (733), TMaskUtils.GetTextWithoutSpaceChar (733)

29.4.3 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 (732), FormatMaskText (733), TMaskUtils (733), TMaskUtils.InputMask (735)

29.5 TMaskUtils

29.5.1 Description

`TEditMask` can be used to work with edit masks. It is used in the FormatMaskInput (732), FormatMaskText (733) and MaskDoFormatText (733) calls to do the actual work.

In general, the work is done by setting the Mask (734) and Values (734) properties, and reading the InputMask (734) property or calling one of ValidateInput (734) or TryValidateInput (734).

See also: Mask (734), Values (734), InputMask (734), ValidateInput (734), TryValidateInput (734)

29.5.2 Method overview

Page	Method	Description
734	TryValidateInput	
734	ValidateInput	Check that the `Value` text is valid for the given mask.

29.5.3 Property overview

Page	Properties	Access	Description
735	InputMask	r	Text to be used as input mask.
734	Mask	rw	The edit mask to use.
734	Value	rw	(input) String value to validate.

29.5.4 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 (734) satisfies the given mask in Mask (734) 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 (734) instead.

Errors: If `Value` does not satisfy the mask, an Exception (730) is raised.

See also: TryValidateInput (734), Value (734), Mask (734), Exception (730)

29.5.5 TMaskUtils.TryValidateInput

Synopsis:

Declaration: `function TryValidateInput(out ValidatedString: string) : Boolean`

Visibility: public

Description: `TryValidateInput` checks that the text in Value (734) satisfies the given mask in Mask (734) and returns `True` if it does, `False` otherwise. It returns `Value` with the mask applied to it in `ValidatedString`.

See also: ValidateInput (734), Value (734), Mask (734)

29.5.6 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 (734).

See also: Value (734), InputMask (735)

29.5.7 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 (734).

See also: Mask (734), InputMask (735)

29.5.8 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 (734), Value (734)

Chapter 30

Reference for unit 'memds'

30.1 Used units

Table 30.1: Used units by unit 'memds'

Name	Page
Classes	??
DB	351
System	??
sysutils	??
Types	??

30.2 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)

30.3 Constants, types and variables

30.3.1 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.

30.4 MDSError

30.4.1 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 (747), TField.DataType (475)

30.5 TMemDataset

30.5.1 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 (409)

30.5.2 Method overview

Page	Method	Description
741	BookmarkValid	Determines if the specified Bookmark is valid.
743	Clear	Clears the content in the in-memory dataset.
741	CompareBookmarks	Gets the relative order for the specified Bookmarks.
745	CopyFromDataset	Loads field definitions and optional data from the specified TDataset.
740	Create	Constructor for the class instance.
741	CreateBlobStream	Creates a stream used to read or write Blob field data in the in-memory dataset.
742	CreateTable	Creates the internal storage for records in the in-memory dataset.
743	DataSize	Size of the internal TMemoryStream used in the in-memory dataset.
740	Destroy	Destructor for the class instance.
744	LoadFromFile	Loads the content for the dataset from the specified file name.
744	LoadFromStream	Loads the content for the dataset from the specified stream.
742	Locate	Locates a record with the specified values in the in-memory dataset.
742	Lookup	Searches for a record with the specified values, and returns a list of values.
743	SaveToFile	Saves field definitions and optional record data to the specified file name.
744	SaveToStream	Saves field definitions and optional record data to the specified stream.

30.5.3 Property overview

Page	Properties	Access	Description
747	Active		Indicates if the in-memory dataset is Active.
749	AfterCancel		
748	AfterClose		
749	AfterDelete		
748	AfterEdit		
748	AfterInsert		
747	AfterOpen		
749	AfterPost		
750	AfterScroll		
749	BeforeCancel		
748	BeforeClose		
749	BeforeDelete		
748	BeforeEdit		
748	BeforeInsert		
747	BeforeOpen		
749	BeforePost		
750	BeforeScroll		
747	FieldDefs		Field definitions for the in-memory dataset.
746	FileModified	r	Indicates if the in-memory dataset has been modified.
746	FileName	rw	File name used to read or write field definitions and optional data.
746	Filter		Filter for the dataset.
747	Filtered		Indicates if records in the dataset are filtered using OnFilterRecord.
750	OnDeleteError		
750	OnEditError		
751	OnFilterRecord		Event handler signalled to include or exclude records in the in-memory dataset.
750	OnNewRecord		
751	OnPostError		

30.5.4 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:

BookmarkSizeSizeOf(LongInt)

30.5.5 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.

30.5.6 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.

30.5.7 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.

30.5.8 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 (493), TDataset.State (437), TField.ReadOnly (486)

30.5.9 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 (426), TLocateOption (363), TLocateOptions (363), TDataset.Resync (429)

30.5.10 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 (742)

30.5.11 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 (439), TMemDataset.Clear (743), TDataset.FieldDefs (434), CreateTable (742)

30.5.12 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.

30.5.13 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 (439), TDataset.Close (418), TDataset.FieldDefs (434)

30.5.14 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.

30.5.15 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 (744)

30.5.16 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 (744)

30.5.17 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 (744), TMemDataset.SaveToFile (743)

30.5.18 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 (437), TDataset.FieldDefs (434), TMemDataset.CreateTable (742), TDataset.Append (416), TDataset.Post (428)

30.5.19 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

30.5.20 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.

Remark

See also: TMemDataset.Filtered (747), TMemDataset.OnFilterRecord (751)

30.5.21 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.

30.5.22 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 (751), TDataset.Filter (438)

30.5.23 TMemDataset.Active

Synopsis: Indicates if the in-memory dataset is Active.

Declaration: `Property Active :`

Visibility: published

Access:

30.5.24 TMemDataset.FieldDefs

Synopsis: Field definitions for the in-memory dataset.

Declaration: `Property FieldDefs :`

Visibility: published

Access:

30.5.25 TMemDataset.BeforeOpen

Synopsis:

Declaration: `Property BeforeOpen :`

Visibility: published

Access:

30.5.26 TMemDataset.AfterOpen

Synopsis:

Declaration: `Property AfterOpen :`

Visibility: published

Access:

30.5.27 TMemDataset.BeforeClose

Synopsis:

Declaration: `Property BeforeClose :`

Visibility: published

Access:

30.5.28 TMemDataset.AfterClose

Synopsis:

Declaration: `Property AfterClose :`

Visibility: published

Access:

30.5.29 TMemDataset.BeforeInsert

Synopsis:

Declaration: `Property BeforeInsert :`

Visibility: published

Access:

30.5.30 TMemDataset.AfterInsert

Synopsis:

Declaration: `Property AfterInsert :`

Visibility: published

Access:

30.5.31 TMemDataset.BeforeEdit

Synopsis:

Declaration: `Property BeforeEdit :`

Visibility: published

Access:

30.5.32 TMemDataset.AfterEdit

Synopsis:

Declaration: `Property AfterEdit :`

Visibility: published

Access:

30.5.33 TMemDataset.BeforePost

Synopsis:

Declaration: `Property BeforePost :`

Visibility: published

Access:

30.5.34 TMemDataset.AfterPost

Synopsis:

Declaration: `Property AfterPost :`

Visibility: published

Access:

30.5.35 TMemDataset.BeforeCancel

Synopsis:

Declaration: `Property BeforeCancel :`

Visibility: published

Access:

30.5.36 TMemDataset.AfterCancel

Synopsis:

Declaration: `Property AfterCancel :`

Visibility: published

Access:

30.5.37 TMemDataset.BeforeDelete

Synopsis:

Declaration: `Property BeforeDelete :`

Visibility: published

Access:

30.5.38 TMemDataset.AfterDelete

Synopsis:

Declaration: `Property AfterDelete :`

Visibility: published

Access:

30.5.39 TMemDataset.BeforeScroll

Synopsis:

Declaration: `Property BeforeScroll :`

Visibility: published

Access:

30.5.40 TMemDataset.AfterScroll

Synopsis:

Declaration: `Property AfterScroll :`

Visibility: published

Access:

30.5.41 TMemDataset.OnDeleteError

Synopsis:

Declaration: `Property OnDeleteError :`

Visibility: published

Access:

30.5.42 TMemDataset.OnEditError

Synopsis:

Declaration: `Property OnEditError :`

Visibility: published

Access:

Description:

See also: (**??**)

30.5.43 TMemDataset.OnNewRecord

Synopsis:

Declaration: `Property OnNewRecord :`

Visibility: published

Access:

30.5.44 TMemDataset.OnPostError

Synopsis:

Declaration: `Property OnPostError :`

Visibility: published

Access:

30.5.45 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 (447), TFilterRecordEvent (362), TDataSet.Filtered (438), TDataSet.Resync (429)

Chapter 31

Reference for unit 'MSSQLConn'

31.1 Used units

Table 31.1: Used units by unit 'MSSQLConn'

Name	Page
BufDataset	142
Classes	??
DB	351
dblib	??
SQLDB	844
System	??
sysutils	??

31.2 Overview

Connector to Microsoft SQL Server databases. Needs FreeTDS dblib library.

31.3 Constants, types and variables

31.3.1 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.

31.4 EMSSQLDatabaseError

31.4.1 Description

Sybase/MS SQL Server specific error.

31.4.2 Property overview

Page	Properties	Access	Description
753	DBErrorCode	r	Sybase/MS SQL Server error code.

31.4.3 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.

31.5 TMSSQLConnection

31.5.1 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')

31.5.2 Method overview

Page	Method	Description
754	Create	Create a new instance of `TMSSQLConnection`.
754	CreateDB	Create a new MS SQL database.
755	DropDB	Drop a MS SQL database.
754	GetConnectionInfo	Return some information about the connection.

31.5.3 Property overview

Page	Properties	Access	Description
756	CharSet		
756	Connected		Is the connection active.
757	DatabaseName		
756	HostName		Host and optionally port or instance.
757	KeepConnection		Keep connection alive.
757	LoginPrompt		Show login prompt.
758	OnLogin		Called when logging in.
757	Params		
755	Password		
756	Role		Role for user.
755	Transaction		Default transaction.
755	UserName		

31.5.4 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.

31.5.5 TMSSQLConnection.GetConnectionInfo

Synopsis: Return some information about the connection.

Declaration:
```
function GetConnectionInfo(InfoType: TConnInfoType) : string;  Override
```

Visibility: public

Description: `GetConnectionInfo` overrides TSQLConnection.GetConnectionInfo (875) to return the relevant information for the Interbase/Firebird connection.

See also: TSQLConnection.GetConnectionInfo (875), TConnInfoType (854)

31.5.6 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 (755)

31.5.7 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 (754)

31.5.8 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).

31.5.9 TMSSQLConnection.Transaction

Synopsis: Default transaction.

Declaration: `Property Transaction :`

Visibility: published

Access:

Description: `Transaction` is redeclared from TSQLConnection.Transaction (752)

See also: TSQLConnection.Transaction (752)

31.5.10 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).

31.5.11 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.

31.5.12 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

31.5.13 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.

31.5.14 TMSSQLConnection.Role

Synopsis: Role for user.

Declaration: `Property Role :`

Visibility: published

Access:

Description: `Role` is redeclared from TSQLConnection.Role (752)

31.5.15 TMSSQLConnection.DatabaseName

Declaration: `Property DatabaseName :`

Visibility: published

Access:

Description: `TMSSQLConnection` specific: the `master` database should always exist on a server.

31.5.16 TMSSQLConnection.KeepConnection

Synopsis: Keep connection alive.

Declaration: `Property KeepConnection :`

Visibility: published

Access:

Description: `KeepConnection` is redeclared from TSQLConnection.KeepConnection (752)

See also: TSQLConnection.KeepConnection (752)

31.5.17 TMSSQLConnection.LoginPrompt

Synopsis: Show login prompt.

Declaration: `Property LoginPrompt :`

Visibility: published

Access:

Description: `LoginPrompt` is redeclared from TSQLConnection.LoginPrompt (752)

See also: TSQLConnection.LoginPrompt (752)

31.5.18 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.

31.5.19 TMSSQLConnection.OnLogin

Synopsis: Called when logging in.

Declaration: `Property OnLogin :`

Visibility: published

Access:

Description: `OnLogin` is redeclared from TSQLConnection.OnLogin (752)

See also: TSQLConnection.OnLogin (752)

31.6 TMSSQLConnectionDef

31.6.1 Description

Describes the MS SQL connection properties for TSQLConnector (752)

See also: TMSSQLConnection (753), TSQLConnector (752)

31.6.2 Method overview

Page	Method	Description
758	ConnectionClass	Connection class to use.
759	DefaultLibraryName	Default name of the MSSQL client library.
759	Description	Short description of connection.
759	LoadedLibraryName	Actally loaded library name.
759	LoadFunction	Return Function to call when loading MS-SQL support.
758	TypeName	Connection type name.
759	UnLoadFunction	Return Function to call when unloading MS-SQL support.

31.6.3 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.

31.6.4 TMSSQLConnectionDef.ConnectionClass

Synopsis: Connection class to use.

Declaration: `class function ConnectionClass : TSQLConnectionClass;  Override`

Visibility: default

Description: `ConnectionClass` returns TMSSQLConnection (753)

See also: TMSSQLConnection (753)

31.6.5 TMSSQLConnectionDef.Description

Synopsis: Short description of connection.

Declaration: `class function Description : string;  Override`

Visibility: default

Description: `Description` describes the MS SQL connector type.

31.6.6 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.

31.6.7 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.

31.6.8 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.

31.6.9 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 (759)

31.7 TSybaseConnection

31.7.1 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)

31.7.2 Method overview

Page	Method	Description
760	Create	Create a Sybase database connection.

31.7.3 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.

31.8 TSybaseConnectionDef

31.8.1 Description

Describes the MS SQL connection properties for TSQLConnector (752)

See also: TSybaseConnection (760), TSQLConnector (752)

31.8.2 Method overview

Page	Method	Description
761	ConnectionClass	Connection class to use.
761	Description	Short description of connection.
760	TypeName	Connection type name.

31.8.3 TSybaseConnectionDef.TypeName

Synopsis: Connection type name.

Declaration: `class function TypeName : string;  Override`

Visibility: default

Description: `TypeName` returns the unique name of the Sybase connection.

31.8.4 TSybaseConnectionDef.ConnectionClass

Synopsis: Connection class to use.

Declaration: `class function ConnectionClass : TSQLConnectionClass;  Override`

Visibility: default

Description: `ConnectionClass` returns TSybaseConnection (760)

See also: TSybaseConnection (760)

31.8.5 TSybaseConnectionDef.Description

Synopsis: Short description of connection.

Declaration: `class function Description : string;  Override`

Visibility: default

Description: `Description` describes the Sybase connector type.

Chapter 32

Reference for unit 'nullstream'

32.1 Used units

Table 32.1: Used units by unit 'nullstream'

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

32.2 Overview

The `nullstream` unit implements TNullStream (762), 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.

32.3 ENullStreamError

32.3.1 Description

`ENullStreamError` is the exception raised when TNullStream.Seek (763) results in an invalid position.

See also: TNullStream.Seek (763)

32.4 TNullStream

32.4.1 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 (763), TNullStream.Write (763), #rtl.classes.TStream.Size (**??**)

32.4.2 Method overview

Page	Method	Description
764	Create	Create a new instance.
763	Read	Read null bytes from the stream.
763	Seek	Set current position in the stream.
763	Write	Write to stream.

32.4.3 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 (763), #rtl.classes.TStream.Size (**??**)

32.4.4 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 (763), #rtl.classes.TStream.Size (**??**)

32.4.5 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 (762) exception is raised.

See also: TNullStream.Read (763), TNullStream.Write (763), #rtl.classes.TStream.Seek (**??**)

32.4.6 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 33

Reference for unit 'Pipes'

33.1 Used units

Table 33.1: Used units by unit 'Pipes'

Name	Page
Classes	??
System	??
sysutils	??

33.2 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.

33.3 Constants, types and variables

33.3.1 Constants

```
ENoSeekMsg = 'Cannot seek on pipes'
```

Constant used in EPipeSeek (766) exception.

```
EPipeMsg = 'Failed to create pipe.'
```

Constant used in EPipeCreation (766) exception.

33.4 Procedures and functions

33.4.1 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 (766)

33.4.2 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 (765), 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 (765) 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 (766) exception is raised.

See also: CreatePipeHandles (765), TInputPipeStream (767), TOutputPipeStream (769)

33.5 EPipeCreation

33.5.1 Description

Exception raised when an error occurred during the creation of a pipe pair.

33.6 EPipeError

33.6.1 Description

Exception raised when an invalid operation is performed on a pipe stream.

33.7 EPipeSeek

33.7.1 Description

Exception raised when an invalid seek operation is attempted on a pipe.

33.8 TInputPipeStream

33.8.1 Description

`TInputPipeStream` is created by the CreatePipeStreams (766) 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 (766), TOutputPipeStream (769)

33.8.2 Method overview

Page	Method	Description
767	Destroy	Destroy this instance of the input pipe stream.
768	Read	Read data from the stream to a buffer.
768	Seek	Set the current position of the stream.
767	Write	Write data to the stream.

33.8.3 Property overview

Page	Properties	Access	Description
768	NumBytesAvailable	r	Number of bytes available for reading.

33.8.4 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 (767)

33.8.5 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 (768), Seek (768)

33.8.6 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 (766) exception is raised if the stream does not allow the requested seek operation.

See also: EPipeSeek (766), Seek (**??**)

33.8.7 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 (767), Seek (768)

33.8.8 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 (768)

33.9 TOutputPipeStream

33.9.1 Description

`TOutputPipeStream` is created by the CreatePipeStreams (766) call to represent the writing end of a pipe. It is a TStream (**??**) descendent which does not allow reading.

See also: TStream (**??**), CreatePipeStreams (766), TInputPipeStream (767)

33.9.2 Method overview

Page	Method	Description
769	Destroy	Destroy this instance of the output pipe stream.
769	Read	Read data from the stream.
769	Seek	Sets the position in the stream.

33.9.3 Property overview

Page	Properties	Access	Description
770	DontClose	rw	

33.9.4 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 (**??**)

33.9.5 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 (766) exception is raised if this method is called.

33.9.6 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 (769)

33.9.7 TOutputPipeStream.DontClose

Declaration: `Property DontClose : Boolean`

Visibility: public

Access: Read,Write

Chapter 34

Reference for unit 'pooledmm'

34.1 Used units

Table 34.1: Used units by unit 'pooledmm'

Name	Page
Classes	??
System	??

34.2 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.

34.3 Constants, types and variables

34.3.1 Types

```
PPooledMemManagerItem = ^TPooledMemManagerItem
```

`PPooledMemManagerItem` is a pointer type, pointing to a TPooledMemManagerItem (772) item, used in a linked list.

```
TEnumItemsMethod = procedure(Item: Pointer) of object
```

`TEnumItemsMethod` is a prototype for the callback used in the TNonFreePooledMemManager.EnumerateItems (773) call. The parameter `Item` will be set to each of the pointers in the item list of TNonFreePooledMemManager (772).

34.4 TPooledMemManagerItem

```
TPooledMemManagerItem = record
```

```
  Next : PPooledMemManagerItem;
end
```

TPooledMemManagerItem is used internally by the TPooledMemManager (774) class to maintain the free list block. It simply points to the next free block.

34.5 TNonFreePooledMemManager

34.5.1 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.

34.5.2 Method overview

Page	Method	Description
772	Clear	Clears the memory.
772	Create	Creates a new instance of TNonFreePooledMemManager.
773	Destroy	Removes the TNonFreePooledMemManager instance from memory.
773	EnumerateItems	Enumerate all items in the list.
773	NewItem	Return a pointer to a new memory block.

34.5.3 Property overview

Page	Properties	Access	Description
773	ItemSize	r	Size of an item in the list.

34.5.4 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 (773) is valid after a call to Clear

See also: NewItem (773)

34.5.5 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 (773)

34.5.6 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 (772), TNonFreePooledMemManager.Clear (772)

34.5.7 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 (773). 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 (772)

34.5.8 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)

34.5.9 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 (772)

34.6 TPooledMemManager

34.6.1 Description

`TPooledMemManager` is a class which maintains a linked list of blocks, represented by the TPooledMemManagerItem (772) record. It should not be used directly, but should be descended from and the descendent should implement the actual memory manager.

See also: TPooledMemManagerItem (772)

34.6.2 Method overview

Page	Method	Description
774	Clear	Clears the list.
774	Create	Creates a new instance of the `TPooledMemManager` class.
775	Destroy	Removes an instance of `TPooledMemManager` class from memory.

34.6.3 Property overview

Page	Properties	Access	Description
776	AllocatedCount	r	Total number of allocated items in the list.
775	Count	r	Number of items in the list.
776	FreeCount	r	Number of free items in the list.
776	FreedCount	r	Total number of freed items in the list.
775	MaximumFreeCountRatio	rw	Maximum ratio of free items over total items.
775	MinimumFreeCount	rw	Minimum count of free items in the list.

34.6.4 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 (776)

34.6.5 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 (775)

34.6.6 TPooledMemManager.Destroy

Synopsis: Removes an instance of `TPooledMemManager` class from memory.

Declaration: `destructor Destroy;  Override`

Visibility: public

Description: `Destroy` calls Clear (774) and then calls the inherited destroy.

`Destroy` should never be called directly. Instead `Free` should be used, or `FreeAndNil`

See also: TPooledMemManager.Create (774)

34.6.7 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 (775)

34.6.8 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 (775)

34.6.9 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 (776), TPooledMemManager.AllocatedCount (776), TPooledMemManager.FreedCount (776)

34.6.10 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 (775), TPooledMemManager.AllocatedCount (776), TPooledMemManager.FreedCount (776)

34.6.11 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 (775), TPooledMemManager.FreeCount (776), TPooledMemManager.FreedCount (776)

34.6.12 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 (775), TPooledMemManager.FreeCount (776), TPooledMemManager.AllocatedCount (776)

Chapter 35

Reference for unit 'process'

35.1 Used units

Table 35.1: Used units by unit 'process'

Name	Page
Classes	??
Math	??
Pipes	765
System	??
sysutils	??

35.2 Overview

The Process unit contains the code for the TProcess (783) 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.

35.3 Constants, types and variables

35.3.1 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 (783) class during the RunCommandLoop (788) 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 (793). 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 35.2: 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 (785), 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 (778).

```
TProcessPriority = (ppHigh,ppIdle,ppNormal,ppRealTime,ppBelowNormal
  ,
                    ppAboveNormal)
```

Table 35.3: 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 (779) 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 35.4: 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 (788) call, reported through the TProcess.OnRunCommandEvent (792) event handler.

```
TRunCommandEventCodeSet = Set of TRunCommandEventCode
```

TRunCommandEventCodeSet is a set of TRunCommandEventCode (779) values.

```
TShowWindowOptions = (swoNone,swoHIDE,swoMaximize,swoMinimize,
  swoRestore,swoShow,swoShowDefault,
  swoShowMaximized,swoShowMinimized,
  swoshowMinNOActive,swoShowNA,swoShowNoActivate,
  swoShowNormal)
```

Table 35.5: 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 35.6: Enumeration values for type TStartupOption

Value	Explanation
suoUseCountChars	Use the console character width as specified in TProcess (783).
suoUseFillAttribute	Use the console fill attribute as specified in TProcess (783).
suoUsePosition	Use the window sizes as specified in TProcess (783).
suoUseShowWindow	Use the Show Window options specified in TShowWindowOption (780).
suoUseSize	Use the window sizes as specified in TProcess (783).

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 (780).

35.3.2 Variables

```
DefaultTProcess : TProcessClass = TPROCESS
```

`DefaultTProcess` is the process class used by the RunCommand (782) and RunCommandInDir (782) 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 (781) function. If XTermProgram (781) 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 (781) is called.

35.4 Procedures and functions

35.4.1 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 (794)

35.4.2 DetectXTerm

Synopsis: Detect the terminal program.

Declaration: `function DetectXTerm : string`

Visibility: default

Description: `DetectXTerm` checks if XTermProgram (781) is set. if so, it returns that. If `XTermProgram` is empty, the list specified in TryTerminals (781) is tested for existence. If none is found, then the `DESKTOP_SESSION` environment variable is examined:

kdekonsole is used if it is found.

gnomegnome-terminal is used if it is found

windowmakeraterm 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 (781), TryTerminals (781), TProcess.XTermProgram (801)

35.4.3 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 (782) 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 (782)

35.4.4 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 (778) 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 (783), RunCommand (782), TProcessOptions (778)

35.5 EProcess

35.5.1 Description

Exception raised when an error occurs in a TProcess routine.

See also: TProcess (783)

35.6 TPROCESS

35.6.1 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 (794) property to the full pathname of the program that should be executed, and call Execute (785). To determine whether the process is still running (i.e. has not stopped executing), the Running (798) property can be checked.

More advanced techniques can be used with the Options (797) settings.

See also: Create (785), Execute (785), Running (798), CommandLine (794), Options (797)

35.6.2 Method overview

Page	Method	Description
786	CloseInput	Close the input stream of the process.
786	CloseOutput	Close the output stream of the process.
786	CloseStderr	Close the error stream of the process.
785	Create	Create a new instance of the `TProcess` class.
785	Destroy	Destroy this instance of `TProcess`.
785	Execute	Execute the program with the given options.
788	ReadInputStream	Read available data from input stream.
786	Resume	Resume execution of a suspended process.
788	RunCommandLoop	Execute command and collect output in strings.
787	Suspend	Suspend a running process.
787	Terminate	Terminate a running process.
787	WaitOnExit	Wait for the program to stop executing.

35.6.3 Property overview

Page	Properties	Access	Description
793	Active	rw	Start or stop the process.
793	ApplicationName	rw	Name of the application to start (deprecated).
794	CommandLine	rw	Command-line to execute (deprecated).
795	ConsoleTitle	rw	Title of the console window.
796	CurrentDirectory	rw	Working directory of the process.
796	Desktop	rw	Desktop on which to start the process.
796	Environment	rw	Environment variables for the new process.
794	Executable	rw	Executable name. Supersedes `CommandLine` and `ApplicationName`.
792	ExitCode	r	Exit code of the process.
791	ExitStatus	r	Exit status of the process.
801	FillAttribute	rw	Color attributes of the characters in the console window (Windows only).
789	Handle	r	Handle of the process.
792	InheritHandles	rw	Should the created process inherit the open handles of the current process.
790	Input	r	Stream connected to standard input of the process.
793	OnForkEvent	rw	Event triggered after fork occurred on Linux.
792	OnRunCommandEvent	rw	Event handler, called when `RunCommandLoop` is executing.
797	Options	rw	Options to be used when starting the process.
790	Output	r	Stream connected to standard output of the process.
795	Parameters	rw	Command-line arguments. Supersedes `CommandLine`.
793	PipeBufferSize	rw	Buffer size to be used when using pipes.
797	Priority	rw	Priority at which the process is running.
789	ProcessHandle	r	Alias for Handle (789).
789	ProcessID	r	ID of the process.
792	RunCommandSleepTime	rw	Sleep time between attempts to collect data.
798	Running	r	Determines whether the process is still running.
799	ShowWindow	rw	Determines how the process main window is shown (Windows only).
798	StartupOptions	rw	Additional (Windows) startup options.
791	Stderr	r	Stream connected to standard diagnostic output of the process.
789	ThreadHandle	r	Main process thread handle.
790	ThreadID	r	ID of the main process thread.
799	WindowColumns	rw	Number of columns in console window (windows only).
799	WindowHeight	rw	Height of the process main window.
800	WindowLeft	rw	X-coordinate of the initial window (Windows only).
788	WindowRect	rw	Positions for the main program window.
800	WindowRows	rw	Number of rows in console window (Windows only).
800	WindowTop	rw	Y-coordinate of the initial window (Windows only).
801	WindowWidth	rw	Height of the process main window (Windows only).
801	XTermProgram	rw	XTerm program to use (UNIX only).

35.6.4 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.

35.6.5 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 (785)

35.6.6 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 (794), applying as much as of the specified options as supported on the current platform.

If the `poWaitOnExit` option is specified in Options (797), 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 (787) call can be used to wait for it to close, or the Running (798) call can be used to check whether it is still running.

The TProcess.Terminate (787) call can be used to terminate the program if it is still running, or the Suspend (787) call can be used to temporarily stop the program's execution.

The ExitStatus (791) function can be used to check the program's exit status, after it has stopped executing.

Errors: On error a EProcess (783) exception is raised.

See also: TProcess.Running (798), TProcess.WaitOnExit (787), TProcess.Terminate (787), TProcess.Suspend (787), TProcess.Resume (786), TProcess.ExitStatus (791), TProcess.ExitCode (792)

35.6.7 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 (790), StdErr (791), Output (790), CloseOutput (786), CloseStdErr (786)

35.6.8 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 (790), Input (790), StdErr (791), CloseInput (786), CloseStdErr (786)

35.6.9 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 (790), Input (790), StdErr (791), CloseInput (786), CloseStdErr (786)

35.6.10 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 (797).

Errors: None.

See also: TProcess.Suspend (787), TProcess.Options (797), TProcess.Execute (785), TProcess.Terminate (787)

35.6.11 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 (786) call.

`Suspend` is fundamentally different from TProcess.Terminate (787) which actually stops the process.

Errors: On error, a nonzero result is returned.

See also: TProcess.Options (797), TProcess.Resume (786), TProcess.Terminate (787), TProcess.Execute (785)

35.6.12 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 (791), TProcess.Suspend (787), TProcess.Execute (785), TProcess.WaitOnExit (787), TProcess.ExitCode (792)

35.6.13 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 (791), TProcess.Terminate (787), TProcess.Running (798), TProcess.ExitCode (792)

35.6.14 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.

35.6.15 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 (792) event handler is called during the various stages of the call. If it is not explicitly set, a sleep period specified by TProcess.RunCommandSleepTime (792) is interjected between the various read calls.

The return value of this call is 1 for error, zero for success.

See also: TProcess.OnRunCommandEvent (792), TProcess.RunCommandSleepTime (792)

35.6.16 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

35.6.17 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 (785) has been called. It is not reset after the process stopped.

See also: TProcess.ThreadHandle (789), TProcess.ProcessID (789), TProcess.ThreadID (790)

35.6.18 TPROCESS.ProcessHandle

Synopsis: Alias for Handle (789).

Declaration: `Property ProcessHandle : THandle`

Visibility: public

Access: Read

Description: `ProcessHandle` equals Handle (789) and is provided for completeness only.

See also: TProcess.Handle (789), TProcess.ThreadHandle (789), TProcess.ProcessID (789), TProcess.ThreadID (790)

35.6.19 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 (785) has been called. It is not reset after the process stopped.

See also: TProcess.Handle (789), TProcess.ProcessID (789), TProcess.ThreadID (790)

35.6.20 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 (785) has been called. It is not reset after the process stopped.

See also: TProcess.Handle (789), TProcess.ThreadHandle (789), TProcess.ThreadID (790)

35.6.21 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 (785) has been called. It is not reset after the process stopped.

See also: TProcess.ProcessID (789), TProcess.Handle (789), TProcess.ThreadHandle (789)

35.6.22 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 (797).

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 (790), TProcess.StdErr (791), TProcess.Options (797), TProcessOption (778)

35.6.23 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 (797).

The `Output` stream also contains any data written to standard diagnostic output (`stderr`) when the `poStdErrToOutPut` flag is used in Options (797).

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 (790), TProcess.StdErr (791), TProcess.Options (797), TProcessOption (778)

35.6.24 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 (797).

The `Output` stream equals the Output (790) when the `poStdErrToOutPut` flag is used in Options (797).

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 (790), TProcess.Output (790), TProcess.Options (797), TProcessOption (778)

35.6.25 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 (798), TProcess.Terminate (787), TProcess.ExitCode (792)

35.6.26 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 (777) 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 (791), TProcess.Running (798), TProcess.WaitOnExit (787), TProcess.Terminate (787)

35.6.27 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 (790), TProcess.Output (790), TProcess.StdErr (791)

35.6.28 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 (788) call and when an exception occurs.

See also: TProcess.RunCommandLoop (788)

35.6.29 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.

35.6.30 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 (790), Input (790), StdErr (791), CloseInput (786), CloseStdErr (786), TProcessForkEvent (778)

35.6.31 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 (765)

35.6.32 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 (785), TProcess.Terminate (787)

35.6.33 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 (794). 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 (794) instead, and leave `ApplicationName` empty.

See also: TProcess.CommandLine (794), TProcess.Executable (794), TProcess.Parameters (795)

35.6.34 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 (794) and TProcess.Parameters (795), 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 (793), TProcess.Executable (794), TProcess.Parameters (795)

35.6.35 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 (795).

`Executable` supersedes the TProcess.CommandLine (794) and TProcess.ApplicationName (793) properties, which have been deprecated. However, if either of `CommandLine` or `ApplicationName` is specified, they will be used instead of `Executable`.

See also: CommandLine (794), ApplicationName (793), Parameters (795)

35.6.36 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 (794).

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 (794), CommandLine (794), ApplicationName (793)

35.6.37 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 (799), TProcess.WindowRows (800)

35.6.38 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 (796)

35.6.39 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 (790), TProcess.Output (790), TProcess.StdErr (791)

35.6.40 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 (797)

35.6.41 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 (785) call is made.

Table 35.7:

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 (778), TProcessOptions (778), TProcess.Priority (797), TProcess.StartUpOptions (798)

35.6.42 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 35.8:

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 35.9:

Priority	Nice value
`ppHigh`	20
`ppIdle`	20
`ppNormal`	0
`ppRealTime`	-20

See also: TProcessPriority (778)

35.6.43 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 35.10:

Priority	Meaning
`suoUseShowWindow`	Use the Show Window options specified in ShowWindow (799)
`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 (801).

See also: TProcess.ShowWindow (799), TProcess.WindowHeight (799), TProcess.WindowWidth (801), TProcess.WindowLeft (800), TProcess.WindowTop (800), TProcess.WindowColumns (799), TProcess.WindowRows (800), TProcess.FillAttribute (801)

35.6.44 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 (787), TProcess.Active (793), TProcess.ExitStatus (791), TProcess.ExitCode (792)

35.6.45 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 35.11:

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

35.6.46 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 (798)

See also: TProcess.WindowHeight (799), TProcess.WindowWidth (801), TProcess.WindowLeft (800), TProcess.WindowTop (800), TProcess.WindowRows (800), TProcess.FillAttribute (801), TProcess.StartupOptions (798)

35.6.47 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 (798)

See also: TProcess.WindowWidth (801), TProcess.WindowLeft (800), TProcess.WindowTop (800), TProcess.WindowColumns (799), TProcess.WindowRows (800), TProcess.FillAttribute (801), TProcess.StartupOptions (798)

35.6.48 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 (798)

See also: TProcess.WindowHeight (799), TProcess.WindowWidth (801), TProcess.WindowTop (800), TProcess.WindowColumns (799), TProcess.WindowRows (800), TProcess.FillAttribute (801), TProcess.StartupOptions (798)

35.6.49 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 (798)

See also: TProcess.WindowHeight (799), TProcess.WindowWidth (801), TProcess.WindowLeft (800), TProcess.WindowTop (800), TProcess.WindowColumns (799), TProcess.FillAttribute (801), TProcess.StartupOptions (798)

35.6.50 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 (798)

See also: TProcess.WindowHeight (799), TProcess.WindowWidth (801), TProcess.WindowLeft (800), TProcess.WindowColumns (799), TProcess.WindowRows (800), TProcess.FillAttribute (801), TProcess.StartupOptions (798)

35.6.51 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 (798)

See also: TProcess.WindowHeight (799), TProcess.WindowLeft (800), TProcess.WindowTop (800), TProcess.WindowColumns (799), TProcess.WindowRows (800), TProcess.FillAttribute (801), TProcess.StartupOptions (798)

35.6.52 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 (799), TProcess.WindowWidth (801), TProcess.WindowLeft (800), TProcess.WindowTop (800), TProcess.WindowColumns (799), TProcess.WindowRows (800), TProcess.StartupOptions (798)

35.6.53 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 (797).

If none is specified, DetectXTerm (781) 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:

kdekonsole is used if it is found.

gnomegnome-terminal is used if it is found

windowmakeraterm 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 (797), DetectXTerm (781)

Chapter 36

Reference for unit 'RttiUtils'

36.1 Used units

Table 36.1: Used units by unit 'RttiUtils'

Name	Page
Classes	??
StrUtils	??
System	??
sysutils	??
TypInfo	??

36.2 Overview

The rttiutils unit is a unit providing simplified access to the RTTI information from published properties using the TPropInfoList (805) 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 (807) class can be used for this. The implementation is based on the apputils unit from `RXLib` by *AO ROSNO* and *Master-Bank*

36.3 Constants, types and variables

36.3.1 Constants

```
sPropNameDelimiter : string = '_'
```

Separator used when constructing section/key names.

36.3.2 Types

```
TEraseSectEvent = procedure(const ASection: string) of object
```

TEraseSectEvent is used by TPropsStorage (807) 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 36.2: 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 (810) property of TPropsStorage (807)

```
TPropStorageOptions = Set of TPropStorageOption
```

TPropStorageOptions is the set of TPropStorageOption used in TPropsStorage.Options (810).

```
TReadStrEvent = function(const ASection: string; const Item: string
  ;
                         const Default: string) : string of
  object
```

TReadStrEvent is used by TPropsStorage (807) 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 (807) 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.

36.3.3 Variables

```
FindGlobalComponentCallBack : TFindComponentEvent
```

FindGlobalComponentCallBack is called by UpdateStoredList (804) 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.

36.4 Procedures and functions

36.4.1 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 (804), TPropsStorage.StoreObjectsProps (809) or TPropsStorage.LoadObjectsProps (809).

See also: ParseStoredItem (804), UpdateStoredList (804), TPropsStorage.StoreObjectsProps (809), TPropsStorage.LoadObjectsProps (809)

36.4.2 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 (804). 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 (804), UpdateStoredList (804), TPropsStorage.StoreObjectsProps (809), TPropsStorage.LoadObjectsProps (809)

36.4.3 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 (804) 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 (803) 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 (804), TPropsStorage.StoreObjectsProps (809), TPropsStorage.LoadObjectsProps (809), FindGlobalComponentCallBack (803)

36.5 TPropInfoList

36.5.1 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 (807)

See also: TPropsStorage (807)

36.5.2 Method overview

Page	Method	Description
806	Contains	Check whether a certain property is included.
805	Create	Create a new instance of `TPropInfoList`.
806	Delete	Delete property information from the list.
805	Destroy	Remove the `TPropInfoList` instance from memory.
806	Find	Retrieve property information based on name.
806	Intersect	Intersect 2 property lists.

36.5.3 Property overview

Page	Properties	Access	Description
807	Count	r	Number of items in the list.
807	Items	r	Indexed access to the property type pointers.

36.5.4 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 (805)

36.5.5 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 (805)

36.5.6 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 (806), Intersect (806)

36.5.7 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 (806), Contains (806)

36.5.8 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 (806) call.

Errors: No checking on the validity of `Index` is performed.

See also: Intersect (806)

36.5.9 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 (806), Contains (806)

36.5.10 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 (807)

36.5.11 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 (807)

36.6 TPropsStorage

36.6.1 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 (805)

36.6.2 Method overview

Page	Method	Description
808	LoadAnyProperty	Load a property value.
809	LoadObjectsProps	Load a list of component properties.
809	LoadProperties	Load a list of properties.
808	StoreAnyProperty	Store a property value.
809	StoreObjectsProps	Store a list of component properties.
808	StoreProperties	Store a list of properties.

36.6.3 Property overview

Page	Properties	Access	Description
810	AObject	rw	Object to load or store properties from.
812	OnEraseSection	rw	Erase a section in storage.
811	OnReadString	rw	Read a string value from storage.
811	OnWriteString	rw	Write a string value to storage.
810	Options	rw	Options to take into account when saving or loading properties from the storage.
811	Prefix	rw	Prefix to use in storage.
811	Section	rw	Section name for storage.

36.6.4 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 (810) property of `TPropsStorage`.

Errors: If the property pointer is invalid or `AObject` is invalid, an exception will be raised.

See also: AObject (810), LoadAnyProperty (808), LoadProperties (809), StoreProperties (808)

36.6.5 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 (810) property of `TPropsStorage`.

Errors: If the property pointer is invalid or `AObject` is invalid, an exception will be raised.

See also: AObject (810), StoreAnyProperty (808), LoadProperties (809), StoreProperties (808)

36.6.6 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 (810) object.

Errors: If an invalid property name is specified, an exception will be raised.

See also: AObject (810), StoreAnyProperty (808), LoadProperties (809), LoadAnyProperty (808)

36.6.7 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 (810) object.

Errors: If an invalid property name is specified, an exception will be raised.

See also: AObject (810), StoreAnyProperty (808), StoreProperties (808), LoadAnyProperty (808)

36.6.8 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 (804) 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 (810) property.

Errors: If an invalid component is specified, an exception will be raised.

See also: UpdateStoredList (804), StoreObjectsProps (809), LoadProperties (809), LoadAnyProperty (808)

36.6.9 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 (804) 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 (810) property.

See also: UpdateStoredList (804), LoadObjectsProps (809), LoadProperties (809), LoadAnyProperty (808)

36.6.10 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 (803), TPropStorageOption (803)

36.6.11 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 (809) or LoadObjectProps (809) will destroy any value that this property might have.

See also: LoadProperties (809), LoadAnyProperty (808), StoreProperties (808), StoreAnyProperty (808), StoreObjectProps (809), LoadObjectProps (809)

36.6.12 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 (811)

36.6.13 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 (811)

36.6.14 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 (811), OnEraseSection (812), TReadStrEvent (803)

36.6.15 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 (811), OnEraseSection (812), TWriteStrEvent (803)

36.6.16 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 (811), OnWriteString (811), TEraseSectEvent (802)

Chapter 37

Reference for unit 'simpleipc'

37.1 Used units

Table 37.1: Used units by unit 'simpleipc'

Name	Page
Classes	??
Contnrs	213
syncobjs	958
System	??
sysutils	??

37.2 Overview

The SimpleIPC unit provides classes to implement a simple, one-way IPC mechanism using string messages. It provides a TSimpleIPCServer (830) component for the server, and a TSimpleIPCClient (827) 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.

37.3 Constants, types and variables

37.3.1 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.

37.3.2 Constants

```
MsgVersion = 1
```

Current version of the messaging protocol.

```
mtString = 1
```

String message type.

```
mtUnknown = 0
```

Unknown message type.

37.3.3 Types

```
TIPCClientCommClass = Class of TIPCClientComm
```

`TIPCClientCommClass` is used by TSimpleIPCClient (827) to decide which kind of communication channel to set up.

```
TIPCMessageOverflowAction = (ipcmoaNone,ipcmoaDiscardOld,
  ipcmoaDiscardNew,ipcmoaError)
```

Table 37.2: 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 (830) 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.

37.3.4 Variables

```
DefaultIPCClientClass : TIPCClientCommClass = Nil
```

`DefaultIPCClientClass` is filled with a class pointer indicating which kind of communication protocol class should be instantiated by the TSimpleIPCClient (827) 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 (830) 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.

37.4 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.

37.5 EIPCError

37.5.1 Description

EIPCError is the exception used by the various classes in the SimpleIPC unit to report errors.

37.6 TIPCClientComm

37.6.1 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 (827) 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 (819) component. The descendent components must always be implemented in pairs.

See also: TSimpleIPCClient (827), TIPCServerComm (819), TSimpleIPCServer (830)

37.6.2 Method overview

Page	Method	Description
817	Connect	Connect to the server.
817	Create	Create a new instance of the `TIPCClientComm`.
817	Disconnect	Disconnect from the server.
818	SendMessage	Send a message.
818	ServerRunning	Check if the server is running.

37.6.3 Property overview

Page	Properties	Access	Description
818	Owner	r	`TSimpleIPCClient` instance for which communication must be handled.

37.6.4 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 (827) instance for which it will handle communication. It can be retrieved later using the Owner (818) property.

See also: Owner (818), TSimpleIPCClient (827)

37.6.5 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 (826) and ServerInstance (829) properties of the owning TSimpleIPCClient (827) instance.

`Connect` is called by the TSimpleIPCClient.Connect (828) call or when the Active (826) 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 (828), Active (826), Disconnect (817)

37.6.6 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 (828) call or when the Active (826) 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 (828), Active (826), Connect (817)

37.6.7 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 (827) component.

See also: ServerID (826), InstanceID (834)

37.6.8 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.

37.6.9 TIPCClientComm.Owner

Synopsis: `TSimpleIPCClient` instance for which communication must be handled.

Declaration:
```
Property Owner : TSimpleIPCClient
```

Visibility: public

Access: Read

Description: `Owner` is the TSimpleIPCClient (827) 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 (827), TIPCClientComm.Create (817)

37.7 TIPCServerComm

37.7.1 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 (830) 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 (816) component. The descendent components must always be implemented in pairs.

See also: TSimpleIPCServer (830), TIPCClientComm (816)

37.7.2 Method overview

Page	Method	Description
819	Create	Create a new instance of the communication handler.
820	PeekMessage	See if a message is available.
820	ReadMessage	Read message from the channel.
819	StartServer	Start the server-side of the communication channel.
820	StopServer	Stop the server side of the communication channel.

37.7.3 Property overview

Page	Properties	Access	Description
821	InstanceID	r	Unique identifier for the communication channel.
821	Owner	r	`TSimpleIPCServer` instance for which to handle transport.

37.7.4 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 (821) property.

See also: Owner (821)

37.7.5 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 (826) property of the TSimpleIPCServer (830) 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 (837)). The arrival of new messages can be acted upon with TSimpleIPCServer.OnMessageQueued (835).

Errors: In case of an error, an EIPCError (816) exception is raised.

See also: TSimpleIPCServer (830), TSimpleIPC.Active (826), TSimpleIPCServer.OnMessageQueued (835), TSimpleIPCServer.ThreadTimeOut (837)

37.7.6 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 (826) property of the TSimpleIPCServer (830) instance is set to `False`.

Errors: In case of an error, an EIPCError (816) exception is raised.

See also: TSimpleIPCServer (830), TSimpleIPC.Active (826)

37.7.7 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 (820) call.

See also: ReadMessage (820)

37.7.8 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 (820) to see if a message is available.

See also: TSimpleIPCServer (830)

37.7.9 TIPCServerComm.Owner

Synopsis: `TSimpleIPCServer` instance for which to handle transport.

Declaration: `Property Owner : TSimpleIPCServer`

Visibility: public

Access: Read

Description: `Owner` refers to the TSimpleIPCServer (830) instance for which this instance of `TSimpleIPCServer` handles the transport. It is specified when the `TIPCServerComm` is created.

See also: TSimpleIPCServer (830)

37.7.10 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.

37.8 TIPCServerMsg

37.8.1 Description

`TIPCServerMsg` is an auxiliary class used in the IPC server class TSimpleIPCServer (830). It keeps the data for 1 message. The set of messages is managed in TIPCServerMsgQueue (823). There should normally be no need to use this class directly.

See also: TIPCServerMsgQueue (823)

37.8.2 Method overview

Page	Method	Description
822	Create	Create a new instance of a server message.
822	Destroy	Destroy an instance of a server message.

37.8.3 Property overview

Page	Properties	Access	Description
822	MsgType	rw	Message type.
823	OwnsStream	rw	Does the message own the stream.
822	Stream	r	Stream to store message data.
823	StringMessage	r	String message sent by client.

37.8.4 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 (822)

37.8.5 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 (822)

37.8.6 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 (822)

37.8.7 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 (822)

37.8.8 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 (822)

37.8.9 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.

37.9 TIPCServerMsgQueue

37.9.1 Description

`TIPCServerMsgQueue` implements a message queue with FIFO characteristics. It has support for a maximum queue length (TIPCServerMsgQueue.MaxCount (825)) and various ways of dealing with overflowing queue (TIPCServerMsgQueue.MaxAction (825))

See also: TIPCServerMsgQueue.MaxCount (825), TIPCServerMsgQueue.MaxAction (825)

37.9.2 Method overview

Page	Method	Description
824	Clear	Clear the message queue.
824	Create	Create a new message queue instance.
824	Destroy	Destroy server message queue instance.
825	Pop	Remove the oldest message from the queue.
824	Push	Add a new message to the queue.

37.9.3 Property overview

Page	Properties	Access	Description
825	Count	r	Number of messages in the queue.
825	MaxAction	rw	Action to take when the number of messages will exceed `MaxCount`.
825	MaxCount	rw	Maximum number of messages in the queue, 0 for unlimited.

37.9.4 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 (825) and TIPCServerMsgQueue.MaxAction (825) with their default values (DefaultIPCMessageQueueLimit (816) and DefaultIPCMessageOverflowAction (815), respectively)

Note that the messages are owned by the queue till they are popped of the queue.

See also: TIPCServerMsgQueue.MaxCount (825), TIPCServerMsgQueue.MaxAction (825), DefaultIPCMessageQueueLimit (816), DefaultIPCMessageOverflowAction (815)

37.9.5 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 (824), TIPCServerMsgQueue.Clear (824)

37.9.6 TIPCServerMsgQueue.Clear

Synopsis: Clear the message queue.

Declaration: `procedure Clear`

Visibility: public

Description: `Clear` discards the remaining messages in the list.

37.9.7 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 (825)) 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 (825) is set to `ipcmoaError`, an exception will be raised.

See also: TIPCServerMsgQueue.MaxCount (825), TIPCServerMsgQueue.MaxAction (825), TIPCServerMsgQueue.Pop (825)

37.9.8 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 (824)

37.9.9 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 (825)

37.9.10 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 (825) is equal to `MaxCount`, the MaxAction (825) property is examined to know what to do.

See also: Count (825), MaxAction (825)

37.9.11 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 (825) equals MaxCount (825) and a new message is put in the queue using Push (824):

Do nothing, just add the message.

Discard the oldest message.

Discard the new message.

See also: TIPCServerMsgQueue.Count (825), TIPCServerMsgQueue.MaxCount (825), TIPCServerMsgQueue.Push (824), TIPCMessageOverflowAction (815)

37.10 TSimpleIPC

37.10.1 Description

`TSimpleIPC` is the common ancestor for the TSimpleIPCServer (830) and TSimpleIPCClient (827) classes. It implements some common properties between client and server.

See also: TSimpleIPCServer (830), TSimpleIPCClient (827)

37.10.2 Property overview

Page	Properties	Access	Description
826	Active	rw	Communication channel active.
826	ServerID	rw	Unique server identification.

37.10.3 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 (826) identification.

See also: ServerID (826)

37.10.4 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.

See also: Active (826)

37.11 TSimpleIPCClient

37.11.1 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 (829) or SendStringMessage (829) calls.

See also: TSimpleIPCServer (830), TSimpleIPC (826), TIPCClientComm (816)

37.11.2 Method overview

Page	Method	Description
828	Connect	Connect to the server.
827	Create	Create a new instance of `TSimpleIPCClient`.
827	Destroy	Remove the `TSimpleIPCClient` instance from memory.
828	Disconnect	Disconnect from the server.
829	SendMessage	Send a message to the server.
829	SendStringMessage	Send a string message to the server.
829	SendStringMessageFmt	Send a formatted string message.
828	ServerRunning	Check if the server is running.

37.11.3 Property overview

Page	Properties	Access	Description
829	ServerInstance	rw	Server instance identification.

37.11.4 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 (827)

37.11.5 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 (827)

37.11.6 TSimpleIPCClient.Connect

Synopsis: Connect to the server.

Declaration: `procedure Connect`

Visibility: public

Description: `Connect` connects to the server indicated in the ServerID (826) and InstanceID (834) properties. `Connect` is called automatically if the Active (826) property is set to `True`.

After a successful call to `Connect`, messages can be sent to the server using SendMessage (829) or SendStringMessage (829).

Calling `Connect` if the connection is already open has no effect.

Errors: If creating the connection fails, an EIPCError (816) exception may be raised.

See also: ServerID (826), InstanceID (827), Active (826), SendMessage (829), SendStringMessage (829), Disconnect (828)

37.11.7 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 (828). `Disconnect` is called automatically if the Active (826) 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 (816) exception may be raised.

See also: Active (826), Connect (828)

37.11.8 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 (826) and InstanceID (834) 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 (828)

37.11.9 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 (816) exception is raised.

See also: Connect (828), SendStringMessage (829)

37.11.10 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 (829) method

Errors: Same as for `SendMessage`.

See also: SendMessage (829), Connect (828), SendStringMessageFmt (829)

37.11.11 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 (829) method

Errors: Same as for `SendMessage`.

See also: SendMessage (829), Connect (828), SendStringMessage (829)

37.11.12 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 (834) property set to `False`, and should match the server's InstanceID (834) property.

See also: ServerID (826), GLobal (834), InstanceID (834)

37.12 TSimpleIPCServer

37.12.1 Description

`TSimpleIPCServer` is the server side of the simple IPC communication protocol. The server program should create a `TSimpleIPCServer` instance, set its ServerID (826) property to a unique name for the system, and then set the Active (826) property to `True` (or call StartServer (831)).

After the server was started, it can check for availability of messages with the PeekMessage (832) call, and read the message with ReadMessage (832).

See also: TSimpleIPCClient (827), TSimpleIPC (826), TIPCServerComm (819)

37.12.2 Method overview

Page	Method	Description
831	Create	Create a new instance of `TSimpleIPCServer`.
831	Destroy	Remove the `TSimpleIPCServer` instance from memory.
832	GetMessageData	Read the data of the last message in a stream.
832	PeekMessage	Check if a client message is available.
832	ReadMessage	Read message from the queue.
831	StartServer	Start the server.
832	StopServer	Stop the server.

37.12.3 Property overview

Page	Properties	Access	Description
834	Global	rw	Is the server reachable to all users or not.
834	InstanceID	r	Instance ID.
836	MaxAction	rw	Action to take when the number of messages will exceed `MaxQueue`.
836	MaxQueue	rw	Maximum number of messages in the queue, 0 for unlimited.
833	Message	r	Last read message.
833	MsgData	r	Last message data.
833	MsgType	r	Last message type.
835	OnMessage	rw	Event triggered when a message arrives.
835	OnMessageError	rw	Event called when a new message has arrived, and the queue is full.
835	OnMessageQueued	rw	Event called when a new message has arrived.
835	OnThreadError	rw	Triggered when a thread reports an error.
833	StringMessage	r	Last message as a string.
837	SynchronizeEvents	rw	Should events be run in the main thread ?
836	Threaded	rw	Is the server running threaded or not ?
834	ThreadError	r	Last thread error.
834	ThreadExecuting	r	Is the message thread currently executing ?
837	ThreadTimeout	rw	Timeout waiting for message.

37.12.4 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 (831)

37.12.5 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 (831)

37.12.6 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 (819) 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 (836) 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 (826) property must be set.

Errors: If an error occurs a EIPCError (816) exception may be raised.

See also: TIPCServerComm (819), Active (826), ServerID (826), StopServer (832), TSimpleIPCServer.Threaded (836)

37.12.7 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 (819) descendent).

See also: TIPCServerComm (819), Active (826), ServerID (826), StartServer (831)

37.12.8 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 (832).

See also: ReadMessage (832)

37.12.9 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 (833) and TSimpleIPCServer.MsgData (833)

It is safe to call this even if a watch thread is started.

See also: TSimpleIPCServer.MsgType (833), TSimpleIPCServer.MsgData (833)

37.12.10 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 (833) 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 (832). It will also not clear the data buffer.

See also: StringMessage (833), MsgData (833), MsgType (833)

37.12.11 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 (832).

See also: GetMessageData (832)

37.12.12 TSimpleIPCServer.Message

Synopsis: Last read message.

Declaration: `Property Message : TIPCServerMsg`

Visibility: public

Access: Read

Description: `Message` is the last read message (using TSimpleIPCServer.ReadMessage (832)) from the message queue.

See also: TSimpleIPCServer.ReadMessage (832)

37.12.13 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 (832).

See also: ReadMessage (832)

37.12.14 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 (833) is better suited to read the data.

This property will contain valid data only after a successful call to ReadMessage (832).

See also: StringMessage (833), ReadMessage (832)

37.12.15 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 (826) property to form the unique server endpoint which a client should use.

See also: ServerID (826), Global (834)

37.12.16 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 (834), TSimpleIPCServer.StartServer (831), TSimpleIPCServer.OnThreadError (835), TSimpleIPCServer.Threaded (836)

37.12.17 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 (834), TSimpleIPCServer.StartServer (831), TSimpleIPCServer.OnThreadError (835), TSimpleIPCServer.Threaded (836)

37.12.18 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 (826), InstanceID (834)

37.12.19 TSimpleIPCServer.OnMessage

Synopsis: Event triggered when a message arrives.

Declaration: `Property OnMessage : TNotifyEvent`

Visibility: published

Access: Read,Write

Description: `OnMessage` is called by ReadMessage (832) when a message has been read. The actual message data can be retrieved with one of the StringMessage (833), MsgData (833) or MsgType (833) properties.

See also: StringMessage (833), MsgData (833), MsgType (833)

37.12.20 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 (832), TSimpleIPCServer.OnMessageError (835)

37.12.21 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 (825) is`ipcmoaError`.

37.12.22 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 (834)

See also: TSimpleIPCServer.ThreadExecuting (834), TSimpleIPCServer.StartServer (831), TSimpleIPCServer.OnThreadError (835), TSimpleIPCServer.ThreadError (834), TSimpleIPCServer.Threaded (836)

37.12.23 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 (825) is equal to `MaxQueue`, the MaxAction (836) property is examined to know what to do.

See also: MaxAction (836)

37.12.24 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 (836) and a new message is put in the queue during PeekMessage (832):

Do nothing, just add the message.

Discard the oldest message.

Discard the new message.

See also: TSimpleIPCServer.MaxQueue (836), TSimpleIPCServer.PeekMessage (832), TIPCMessageOverflowAction (815)

37.12.25 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 (831). Trying to set it when the server is started will result in an error.

See also: StartServer (831)

37.12.26 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 (831), TSimpleIPCServer.StopServer (832)

37.12.27 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 (836), StartServer (831)

Chapter 38

Reference for unit 'singleinstance'

38.1 Used units

Table 38.1: Used units by unit 'singleinstance'

Name	Page
Classes	??
System	??
sysutils	??

38.2 Overview

`Singleinstance` contains the basic abstract definition of a class TBaseSingleInstance (839) 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 (838) 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.

38.3 Constants, types and variables

38.3.1 Types

```
TBaseSingleInstanceClass = Class of TBaseSingleInstance
```

`TBaseSingleInstanceClass` is the class of TBaseSingleInstance (839). It is also the type of the DefaultSingleInstanceClass (839) variable.

```
TSingleInstanceParamsEvent = procedure(Sender: TBaseSingleInstance
  ;
```

```
                                        Params: TStringList) of
  object
```

This callback type is the signature of the TBaseSingleInstance.OnServerReceivedParams (842) 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 38.2: 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.

38.3.2 Variables

```
DefaultSingleInstanceClass : TBaseSingleInstanceClass = Nil
```

`DefaultSingleInstanceClass` can be set to a class that implements the abstract methods of TBaseSingleInstance (839). It can be used by applications to instantiate a class without referring directly to that class.

38.4 ESingleInstance

38.4.1 Description

`ESingleInstance` is used to report errors in the single instance implementation(s).

38.5 TBaseSingleInstance

38.5.1 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 (96) class.

See also: TAdvancedSingleInstance (96), advancedsingleinstance (95)

38.5.2 Method overview

Page	Method	Description
841	ClientPostParams	Send parameters to the server.
840	Create	Create a new instance of the `TBaseSingleInstance` class.
840	Destroy	Stop waiting and destroy the instance.
841	ServerCheckMessages	Check for messages from a client application.
840	Start	Check if another instance of the application is running.
841	Stop	Stop the current instance.

38.5.3 Property overview

Page	Properties	Access	Description
843	IsClient	r	Is this application the client instance ?
843	IsServer	r	Is this application the server instance ?
842	OnServerReceivedParams	rw	Event triggered during `ServerCheckMessages`.
842	StartResult	r	Contains the result of the last `Start` call.
842	TimeOutMessages	rw	Timeout when waiting for messages.
842	TimeOutWaitForInstances	rw	Timeout when trying to contact server instance.

38.5.4 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 (842) and TimeOutWaitForInstances (842)

See also: TBaseSingleInstance.Destroy (840), TimeOutMessages (842), TimeOutWaitForInstances (842)

38.5.5 TBaseSingleInstance.Destroy

Synopsis: Stop waiting and destroy the instance.

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

Visibility: public

Description: `Destroy` calls Stop (841) and calls the inherited destructor.

See also: Stop (841)

38.5.6 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 (842)

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 (842), TSingleInstanceStart (839)

38.5.7 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 (840), 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 (840)

38.5.8 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 (841)). This is part of themechanism to communicate parameters from the client to the server. If there are any messages, then the OnServerReceivedParams (842) event will be triggered once for any client that sent parameters.

See also: TBaseSingleInstance.ClientPostParams (841)

38.5.9 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 (839) class.

See also: TBaseSingleInstance.ServerCheckMessages (841)

38.5.10 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 (841) when checking for messages sent by client applications.

See also: ServerCheckMessages (841), TimeOutWaitForInstances (842)

38.5.11 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 (840) call when checking for another instance of the application.

See also: Start (840), TimeOutMessages (842)

38.5.12 TBaseSingleInstance.OnServerReceivedParams

Synopsis: Event triggered during `ServerCheckMessages`.

Declaration: `Property OnServerReceivedParams : TSingleInstanceParamsEvent`

Visibility: public

Access: Read,Write

Description: `OnServerReceivedParams` is the event triggered when ServerCheckMessages (841) 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.

38.5.13 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 (840) call.

See also: Start (840)

38.5.14 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 (842) equals `siServer`.

See also: IsClient (843), Start (840), StartResult (842)

38.5.15 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 (842) equals `siClient`.

See also: IsServer (843), Start (840), StartResult (842)

Chapter 39

Reference for unit 'SQLDB'

39.1 Used units

Table 39.1: Used units by unit 'SQLDB'

Name	Page
BufDataset	142
Classes	??
DB	351
sqlscript	??
SQLTypes	920
System	??
sysutils	??

39.2 Overview

The SQLDB unit defines four main classes to handle data in SQL based databases.

1. TSQLConnection (871) 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 (886) is a #fcl.db.TDataset (409) 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 (915) 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 (902) 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 (845).

39.3 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 (409) 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 (871) 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 (915) 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 (886) 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 (886) can only execute 1 SQL statement during ExecSQL. If many SQL statements must be executed, TSQLScript (902) must be used.

There is much more to `TSQLQuery` than explained here: it can use parameters (see UsingParams (849)) and it can automatically update the data that you edit in it (see UpdateSQLs (848)).

See also: TSQLConnection (871), TSQLTransaction (915), TSQLQuery (886), TSQLConnector (882), TSQLScript (902), UsingParams (849), UpdateSQLs (848)

39.4 Using the universal TSQLConnector type.

The normal procedure when using SQLDB is to use one of the TSQLConnection (871) 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 (882) 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 (883) 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 (858) 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 (882) 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 (871), TSQLConnector (882)

39.5 Retrieving Schema Information.

Schema Information (lists of available database objects) can be retrieved using some specialized calls in TSQLConnection (871):

- TSQLConnection.GetTableNames (874) retrieves a list of available tables. The system tables can be requested.
- TSQLConnection.GetProcedureNames (874) retrieves a list of available stored procedures.
- TSQLConnection.GetFieldNames (875) 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 (865) 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 (874).

stSysTables Retrieves the list of system Tables in database. This is used internally by TSQLConnection.GetTableNames (874) when the system tables are requested

stProcedures Retrieves a list of stored procedures in database. This is used internally by TSQLConnection.GetProcedureNames (874).

stColumns Retrieves the list of columns (fields) in a table. This is used internally by TSQLConnection.GetFieldNames (875).

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).

39.6 Automatic generation of update SQL statements.

SQLDB (more in particular, TSQLQuery (886)) 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 (899) 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 (486) 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 (899) 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 (895), InsertSQL (895) and DeleteSQL (896) 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 (849), TSQLQuery (886), UpdateSQL (895), InsertSQL (895), DeleteSQL (895)

39.7 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 (895), TSQLQuery.DeleteSQL (896), TSQLQuery.InsertSQL (895) properties of TSQLQuery (886) 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 (897), UpdateSQLs (848)

39.8 Constants, types and variables

39.8.1 Constants

```
DefaultMacroChar = '%'
```

`DefaultMacroChar` is the default macro delimiter to use in TSQLQuery (886)

```
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 (871) 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 (880).

```
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 (920).

```
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.

39.8.2 Types

```
TCommitRollbackAction = (caNone,caCommit,caCommitRetaining,caRollback
  ,
                          caRollbackRetaining)
```

Table 39.2: 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 (858) call to register a new TConnectionDef (860) instance.

```
TConnInfoType = (citAll,citServerType,citServerVersion,
  citServerVersionString,citClientName,citClientVersion)
```

Table 39.3: 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 39.4: 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 (871) through the TSQLConnection.OnLog (879) event.

```
TDBEventTypes = sqltypes.TDBEventTypes
```

`TDBEventTypes` is a set of TDBEventType (855) values, which is used to filter the set of event messages that should be sent. The TSQLConnection.LogEvents (880) 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 (879) event handler and for the global GlobalDBLogHook (857) event handling hook. `Sender` will contain the TSQLConnection (871) 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 (855)

```
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 (865) 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 (883)

```
TSQLConnectionOption = (scoExplicitConnect,
  scoApplyUpdatesChecksRowsAffected)
```

Table 39.5: 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 (871) instances using TSQLConnection.Options (880)

```
TSQLConnectionOptions = Set of TSQLConnectionOption
```

Set of TSQLConnectionOption.

```
TSQLQueryOption = (sqoKeepOpenOnCommit,sqoAutoApplyUpdates,
  sqoAutoCommit,sqoCancelUpdatesOnRefresh,
  sqoRefreshUsingSelect,sqoNoCloseOnSQLChange)
```

Table 39.6: 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 (886) instance.

```
TSQLQueryOptions = Set of TSQLQueryOption
```

TSQLQueryOptions is the type of the TSQLQuery.Options (897) property.

```
TSQLSequenceApplyEvent = (saeOnNewRecord,saeOnPost)
```

Table 39.7: 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 39.8: 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 (915).

```
TSQLTransactionOptions = Set of TSQLTransactionOption
```

TSQLTransactionOptions is the property type of TSQLTransaction.Options (919).

```
TStatementType = sqltypes.TStatementType
```

TStatementType describes the kind of SQL statement that was entered in the SQL property of a TSQLQuery (886) component.

39.8.3 Variables

```
GlobalDBLogHook : TDBLogNotifyEvent
```

GlobalDBLogHook can be set in addition to local TSQLConnection.Onlog (879) 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.

39.9 Procedures and functions

39.9.1 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 (858), TConnectionDef (860), TConnectionDef.TypeName (860)

39.9.2 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 (858). The names are the names as returned by TConnectionDef.TypeName (860)

See also: RegisterConnection (858), TConnectionDef.TypeName (860)

39.9.3 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 (860) descendent to register the connection type described in the TConnectionDef (860) descendent. The connection type is registered with the name as returned by TConnectionDef.TypeName (860).

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 (859).

Errors: if `Def` is `Nil`, access violations will occur.

See also: TConnectionDef (860), UnRegisterConnection (859)

39.9.4 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 (860) 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 (860), RegisterConnection (858)

39.10 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 (911) and TSQLQuery (886) 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.

39.11 ESQLDatabaseError

39.11.1 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 (844), ESQLDatabaseError.SQLState (**??**), ESQLDatabaseError.ErrorCode (**??**)

39.11.2 Method overview

Page	Method	Description
859	CreateFmt	Create a new instance of `ESQLDatabaseError`.

39.11.3 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 (**??**)

39.12 TConnectionDef

39.12.1 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 (858). A descendent class should override at least the TConnectionDef.TypeName (860) and TConnectionDef.ConnectionClass (861) methods to return the specific name and connection class to use.

See also: TConnectionDef.TypeName (860), TConnectionDef.ConnectionClass (861), RegisterConnection (858)

39.12.2 Method overview

Page	Method	Description
862	ApplyParams	Apply parameters to an instance of `TSQLConnection`.
861	ConnectionClass	Class to instantiate when this connection is requested.
861	DefaultLibraryName	Default library name.
861	Description	A descriptive text for this connection type.
862	LoadedLibraryName	Currently loaded library.
861	LoadFunction	Return a function to call when the client library must be loaded.
860	TypeName	Name of the connection type.
862	UnLoadFunction	Return a function to call when the client library must be unloaded.

39.12.3 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 (883) property should be set to select this connection type for the universal connection, and is the name that the GetConnectionDef (858) 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 (858)

This name can be an arbitrary name, no restrictions on the allowed characters exist.

See also: TSQLConnector.ConnectorType (883), GetConnectionDef (858), GetConnectionList (858), TConnectionDef.ConnectionClass (861)

39.12.4 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 (860)

39.12.5 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 (860)

39.12.6 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 (855), TConnectionDef (860), TLibraryUnLoadFunction (855)

39.12.7 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 (855), TConnectionDef.UnLoadFunction (862), TConnectionDef.DefaultLibraryName (861), TConnectionDef.LoadedLibraryName (862)

39.12.8 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 (855), TConnectionDef.LoadFunction (861), TConnectionDef.DefaultLibraryName (861), TConnectionDef.LoadedLibraryName (862)

39.12.9 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 (855), TLibraryUnLoadFunction (855), TConnectionDef.LoadFunction (861), TConnectionDef.UnLoadFunction (862), TConnectionDef.DefaultLibraryName (861)

39.12.10 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 (871) 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 (861). `Params` contains the contents of the TSQLConnection.Params (882) property of the connector.

See also: TSQLConnection.Params (882)

39.13 TCustomSQLQuery

39.13.1 Description

`TCustomSQLQuery` encapsulates a SQL statement: it implements all the necessary #fcl.db.TDataset (409) 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 (864) method.

Do not instantiate a `TCustomSQLQuery` class directly, instead use the TSQLQuery (886) descendent.

See also: TSQLQuery (886)

39.13.2 Method overview

Page	Method	Description
866	ApplyUpdates	Apply updates and check result.
863	Create	Create a new instance of `TCustomSQLQuery`.
866	Delete	Delete and optionally apply updates.
863	Destroy	Destroy instance of `TCustomSQLQuery`.
864	ExecSQL	Execute a SQL statement that does not return a result set.
866	MacroByName	Convenience for Macros.ParamByName.
865	ParamByName	Return parameter by name.
866	Post	Post pending changes and optionally apply updates.
864	Prepare	Prepare a query for execution.
865	RowsAffected	Return the number of rows (records) affected by the last DML/DDL statement.
865	SetSchemaInfo	SetSchemaInfo prepares the dataset to retrieve schema info.
864	UnPrepare	Unprepare a prepared query.

39.13.3 Property overview

Page	Properties	Access	Description
867	Prepared	r	Is the query prepared ?
867	SQLConnection	rw	Database as TSQLConnection.
867	SQLTransaction	rw	Transaction as TSQLTransaction.

39.13.4 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 (863)

39.13.5 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 (863)

39.13.6 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 (889), TCustomSQLQuery.UnPrepare (864), TCustomSQLQuery.ExecSQL (864)

39.13.7 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 (894) 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 (889), TCustomSQLQuery.Prepare (864), TCustomSQLQuery.ExecSQL (864)

39.13.8 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 (894), 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 (428) method must be used.

The SQL property should be a single SQL command. To execute multiple SQL statements, use the TSQLScript (902) component instead.

If the statement is a `DML` statement, the number of deleted/updated/inserted rows can be determined using TCustomSQLQuery.RowsAffected (865).

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 (865), TDataset.Open (428)

39.13.9 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 (874) and TSQLConnection.GetProcedureNames (874) to get the necessary schema information from the database.

See also: TSQLConnection.GetTableNames (874), TSQLConnection.GetProcedureNames (874), RetrievingSchemaInformation (848)

39.13.10 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 (864).

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 (864), TSQLConnection (871)

39.13.11 TCustomSQLQuery.ParamByName

Synopsis: Return parameter by name.

Declaration:
```
function ParamByName(const AParamName: string) : TParam
```

Visibility: public

Description: `ParamByName` is a shortcut for Params.ParamByName (547). 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 (547), TSQLQuery.Params (897)

39.13.12 TCustomSQLQuery.MacroByName

Synopsis: Convenience for Macros.ParamByName.

Declaration: `function MacroByName(const AParamName: string) : TParam`

Visibility: public

Description: `MacroByName` checks Macros (862) 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 (547)

39.13.13 TCustomSQLQuery.ApplyUpdates

Synopsis: Apply updates and check result.

Declaration: `procedure ApplyUpdates(MaxErrors: Integer); Override; Overload`

Visibility: public

Description: `ApplyUpdates` is overridden in TCustomSQLQuery (862) to check the result of the update (using RowsAffected (865)).

See also: TSQLQueryOptions (857), TSQLQuery.Options (897)

39.13.14 TCustomSQLQuery.Post

Synopsis: Post pending changes and optionally apply updates.

Declaration: `procedure Post; Override`

Visibility: public

Description: `Post` is overridden from DB.TDataset.Post (844) to implement the auto-applyupdates mechanism: if TSQLQuery.Options (897) contains `sqoAutoApplyUpdates`, then `ApplyUpdates` is called as the last step of the `Post` operation.

See also: TSQLQuery.Options (897), TCustomSQLQuery.ApplyUpdates (866)

39.13.15 TCustomSQLQuery.Delete

Synopsis: Delete and optionally apply updates.

Declaration: `procedure Delete; Override`

Visibility: public

Description: `Delete` is overridden from DB.TDataset.Delete (844) to implement the auto-applyupdates mechanism: if TSQLQuery.Options (897) contains `sqoAutoApplyUpdates`, then `ApplyUpdates` is called as the last step of the `Post` operation.

See also: TSQLQuery.Options (897), TCustomSQLQuery.ApplyUpdates (866)

39.13.16 TCustomSQLQuery.Prepared

Synopsis: Is the query prepared ?

Declaration: `Property Prepared : Boolean`

Visibility: public

Access: Read

Description: `Prepared` is true if Prepare (864) was called for this query, and an UnPrepare (864) 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 (864), TCustomSQLQuery.UnPrepare (864)

39.13.17 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 (871) descendent.

See also: TSQLConnection (871), TCustomSQLQuery.SQLTransaction (867)

39.13.18 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 (915) descendent.

See also: TSQLConnection (871), TCustomSQLQuery.SQLConnection (867)

39.14 TCustomSQLStatement

39.14.1 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 (844) 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 (886) in that it supports transactions (in the Transaction (878) property) and parameters (in the Params (882) property) and as such is a more versatile tool than executing queries using TSQLConnection.ExecuteDirect (873).

To use a TCustomSQLStatement is simple and similar to the use of TSQLQuery (886): set the Database (912) property to an existing connection component, and set the Transaction (914) property. After setting the SQL (914) property and filling Params (913), the SQL statement can be executed with the Execute (869) method.

TCustomSQLStatement is a parent class. Many of the properties are only made public (or published) in the TSQLStatement (911) class, which should be instantiated instead of the TCustomSQLStatement class.

See also: TSQLStatement (911), TDataset (844), TSQLQuery (886), TSQLStatement.Transaction (914), TSQLStatement.Params (913), TCustomSQLStatement.Execute (869), TSQLStatement.Database (912), TSQLConnection.ExecuteDirect (873)

39.14.2 Method overview

Page	Method	Description
868	Create	Create a new instance of TCustomSQLStatement.
868	Destroy	Destroy a TCustomSQLStatement instance.
869	Execute	Execute the SQL statement.
870	ParamByName	Find a parameter by name.
869	Prepare	Prepare the statement for execution.
870	RowsAffected	Number of rows affected by the SQL statement.
869	Unprepare	Unprepare a previously prepared statement.

39.14.3 Property overview

Page	Properties	Access	Description
870	Prepared	r	Is the statement prepared or not.

39.14.4 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 (914)Params (913), ParamCheck (912) and ParseSQL (914) to their initial values.

See also: TSQLStatement.SQL (914), TSQLStatement.Params (913), TSQLStatement.ParamCheck (912), TSQLStatement.ParseSQL (914), TSQLStatement.Destroy (911)

39.14.5 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 (912), TSQLStatement.Transaction (914)

39.14.6 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 (869) 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 (914) statement set or the Database (912) or Transaction (914) properties are empty.

See also: TSQLStatement.SQL (914), TSQLStatement.Database (912), TSQLStatement.Transaction (914), TCustomSQLStatement.Execute (869)

39.14.7 TCustomSQLStatement.Execute

Synopsis: Execute the SQL statement.

Declaration: `procedure Execute`

Visibility: public

Description: `Execute` executes the SQL (914) 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 (914) statement set or the Database (912) or Transaction (914) 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 (914), TSQLStatement.Database (912), TSQLStatement.Transaction (914)

39.14.8 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 (914), TSQLStatement.Database (912), TSQLStatement.Transaction (914), TCustomSQLStatement.Prepare (869)

39.14.9 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 (913) property.

Errors: If no parameter with the given name is found, an exception is raised.

See also: TSQLStatement.Params (913), TParams.ParamByname (844)

39.14.10 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 (869) was called. Not all databases may support this.

See also: TCustomSQLStatement.Execute (869)

39.14.11 TCustomSQLStatement.Prepared

Synopsis: Is the statement prepared or not.

Declaration: `Property Prepared : Boolean`

Visibility: public

Access: Read

Description: `Prepared` equals `True` if Prepare (869) was called (implicitly or explicitly), it returns `False` if not. It can be set to `True` or `False` to call Prepare (869) or UnPrepare (869), respectively.

See also: TCustomSQLStatement.Prepare (869), TCustomSQLStatement.UnPrepare (869)

39.15 TServerIndexDefs

39.15.1 Description

`TServerIndexDefs` is a simple descendent of TIndexDefs (512) that implements the necessary methods to update the list of definitions using the TSQLConnection (871). It should not be used directly.

See also: TSQLConnection (871)

39.15.2 Method overview

Page	Method	Description
871	Create	Create a new instance of `TServerIndexDefs`.
871	Update	Updates the list of indexes.

39.15.3 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 (862) descendent.

Errors: An `EDatabaseError` exception will be raised if `ADataset` is not a TCustomSQLQuery (862) descendent.

39.15.4 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 (871) methods for this.

39.16 TSQLConnection

39.16.1 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 (886), TSQLTransaction (915)

39.16.2 Method overview

Page	Method	Description
872	Create	Create a new instance of `TSQLConnection`.
876	CreateDB	Create a new Database on the server.
872	Destroy	Destroys the instance of the connection.
876	DropDB	Procedure to drop or remove a Database.
873	EndTransaction	End the Transaction associated with this connection.
873	ExecuteDirect	Execute a piece of SQL code directly, using a Transaction if specified.
875	GetConnectionInfo	Return some information about the connection.
875	GetFieldNames	Gets a list of the field names in the specified table.
876	GetNextValue	Get next value for a sequence.
874	GetObjectNames	Return a collection of object names for a given type of object.
874	GetProcedureNames	Gets a list of Stored Procedures in the Database.
875	GetSchemaNames	Get database schema names.
875	GetSequenceNames	Return a list of sequence names.
876	GetStatementInfo	Get statement information.
874	GetTableNames	Get a list of the tables in the specified database.
874	HasTable	
873	StartTransaction	Start the Transaction associated with this Connection.

39.16.3 Property overview

Page	Properties	Access	Description
879	CharSet	rw	The character set to be used in this database.
881	Connected		Is a connection to the server active or not.
877	ConnOptions	r	The set of Connection options being used in the Connection.
881	DatabaseName		The name of the database to which connection is required.
877	FieldNameQuoteChars	rw	Characters used to quote field names.
877	Handle	r	Low level handle used by the connection.
879	HostName	rw	The name of the host computer where the database resides.
881	KeepConnection		Attempt to keep the connection open once it is established.
880	LogEvents	rw	Filter for events to log.
882	LoginPrompt		Should SQLDB prompt for user credentials when a connection is activated.
879	OnLog	rw	Event handler for logging events.
882	OnLogin		Event handler for login process.
880	Options	rw	Options to observe for this connection.
882	Params		Extra connection parameters.
878	Password	rw	Password used when authenticating on the database server.
880	Role	rw	Role in which the user is connecting to the database.
878	Transaction	rw	Default transaction to be used for this connection.
878	UserName	rw	The username for authentication on the database server.

39.16.4 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 (871). After calling the inherited constructor, it will initialize the FieldNameQuoteChars (877) property and some other fields for internal use.

See also: FieldNameQuoteChars (877)

39.16.5 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.

39.16.6 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 (878)). It is equivalent to

```
Connection.Transaction.StartTransaction
```

Errors: If no transaction is assigned, an exception will be raised.

See also: EndTransaction (873)

39.16.7 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 (878)). It is equivalent to

```
Connection.Transaction.EndTransaction
```

Errors: If no transaction is assigned, an exception will be raised.

See also: StartTransaction (873)

39.16.8 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 (886) component and its ExecSQL (864) method.

Errors: If no transaction is assigned, and no transaction is passed, an exception will be raised.

See also: TSQLQuery (886), ExecSQL (864)

39.16.9 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 (923)

39.16.10 TSQLConnection.HasTable

Declaration:
```
function HasTable(const aTable: string; SearchSystemTables: Boolean)
                  : Boolean
```

Visibility: public

39.16.11 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 (874), TSQLConnection.GetFieldNames (875)

39.16.12 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 (874), TSQLConnection.GetFieldNames (875)

39.16.13 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 (874), TSQLConnection.GetProcedureNames (874)

39.16.14 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 (874), TSQLConnection.GetProcedureNames (874), TSQLConnection.GetFieldNames (875)

39.16.15 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.

39.16.16 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 (854)

39.16.17 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 (859)

39.16.18 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 (881) property, the user credentials are taken from the TSQLConnection.UserName (878) and TSQLConnection.Password (878) 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 (876)

39.16.19 TSQLConnection.DropDB

Synopsis: Procedure to drop or remove a Database.

Declaration:
```
procedure DropDB; Virtual
```

Visibility: public

Description: `DropDB` does the opposite of CreateDB (876). 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 (876)

39.16.20 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 (875)

39.16.21 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 (854)

39.16.22 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.

39.16.23 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 (871) descendent to the default for that particular connection type.

See also: TSQLConnection (871)

39.16.24 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 (878) 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 (878), TSQLConnection.HostName (879)

39.16.25 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 (915) 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 (874)

See also: TSQLTransaction (915)

39.16.26 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 (878), TSQLConnection.HostName (879), TSQLConnection.Role (880), TSQLConnection.Charset (879)

39.16.27 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 (878), TSQLConnection.HostName (879), TSQLConnection.UserName (878), TSQLConnection.Role (880)

39.16.28 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 (880), TSQLConnection.Password (878), TSQLConnection.UserName (878), TSQLConnection.DatabaseName (881), TSQLConnection.Charset (879)

39.16.29 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 (880) 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 (857) event handler.

See also: GlobalDBLogHook (857), TSQLConnection.LogEvents (880)

39.16.30 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 (879) and GlobalDBLogHook (857) event handlers. Only event types that are listed in this property will be sent.

See also: GlobalDBLogHook (857), TSQLConnection.OnLog (879)

39.16.31 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:

scoExplicitConnectWhen set, the connection must be explicitly made. Default behaviour is for `TSQLQuery` to implicitly open the connection as needed.

scoApplyUpdatesChecksRowsAffectedWhen set, whenever an update SQL Statement is executed during ApplyOptions of a dataset, the RowsAffected (870) is checked and must be equal to 1.

See also: TCustomSQLQuery.ApplyUpdates (866), TCustomSQLStatement.RowsAffected (870), TCustomSQLQuery.RowsAffected (865)

39.16.32 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 (878), TSQLConnection.UserName (878), TSQLConnection.DatabaseName (881), TSQLConnection.Hostname (879)

39.16.33 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 (881) on host HostName (879) using the credentials specified in UserName (878) and Password (878). If the connection or authentication fails, an exception is raised. This has the same effect as calling Open (396).

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 (844)

See also: TSQLConnection.Password (878), TSQLConnection.UserName (878), TSQLConnection.DatabaseName (881), TSQLConnection.Role (880)

39.16.34 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 (878), TSQLConnection.UserName (878), TSQLConnection.Charset (879), TSQLConnection.Hostname (879)

39.16.35 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.

39.16.36 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 (882) event handler. The UserName (878) and Password (878) properties are ignored in this case.

See also: TSQLConnection.Password (878), TSQLConnection.UserName (878), OnLogin (882)

39.16.37 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 (878), TSQLConnection.UserName (878), TSQLConnection.Hostname (879), TSQLConnection.DatabaseName (881)

39.16.38 TSQLConnection.OnLogin

Synopsis: Event handler for login process.

Declaration: `Property OnLogin :`

Visibility: published

Access:

Description: `OnLogin` will be used when loginPrompt (882) 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 (882)

39.17 TSQLConnector

39.17.1 Description

`TSQLConnector` implements a general connection type. When switching database backends, the normal procedure is to replace one instance of TSQLConnection (871) descendent with another, and connect all instances of TSQLQuery (886) and TSQLTransaction (915) to the new connection.

Using `TSQLConnector` avoids this: the type of connection can be set using the ConnectorType (883) property, which is a string property. The `TSQLConnector` class will (in the background) create the correct TSQLConnection (871) 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 (883), UniversalConnectors (847)

39.17.2 Property overview

Page	Properties	Access	Description
883	ConnectorType	rw	Name of the connection type to use.
883	Port		

39.17.3 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 (858) 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 (858)

39.17.4 TSQLConnector.Port

Declaration: `Property Port :`

Visibility: published

Access:

39.18 TSQLCursor

39.18.1 Description

TSQLCursor is an abstract internal object representing a result set returned by a single SQL select statement (TSQLHandle (885)). statement. It is used by the TSQLQuery (886) component to handle result sets returned by SQL statements.

This object must not be used directly.

See also: TSQLQuery (886), TSQLHandle (885)

39.19 TSQLDBFieldDef

39.19.1 Description

`TSQLDBFieldDef` is a SQLDB specific db.TFieldDef (844) descendent which has room for storing engine-specific data for the result set fields.

See also: TSQLDBFieldDef.SQLDBData (884)

39.19.2 Property overview

Page	Properties	Access	Description
884	SQLDBData	rw	Pointer to store engine-specific data for the result field.

39.19.3 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 (871) descendents to store additional data about fields in a result set. It is not used by `TSQLQuery` itself.

See also: TSQLQuery (886)

39.20 TSQLDBFieldDefs

39.20.1 Description

`TSQLDBFieldDefs` is a `TFieldDefs` descendent which creates TSQLDBFieldDef (884) descendents when a new field is added to the field set, to provide storage for engine-specific field data.

See also: TSQLDBFieldDef (884)

39.21 TSQLDBParam

39.21.1 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 (885) field. This is useful when constructing update or insert queries. It can also store private data needed for the various TSQLConnection (871) descendents in the TSQLDBParam.SQLDBData (885) property.

See also: TSQLDBFieldDef (884), TSQLDBParam.FieldDef (885), TSQLDBParam.SQLDBData (885)

39.21.2 Property overview

Page	Properties	Access	Description
885	FieldDef	rw	Field definition for update SQL.
885	SQLDBData	rw	Private data for TSQLDB descendents.

39.21.3 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 (884), TSQLDBParam.SQLDBData (885)

39.21.4 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 (871) descendents.

See also: TSQLConnection (871), TSQLDBParam.FieldDef (885)

39.22 TSQLDBParams

39.22.1 Description

`TSQLDBParams` is a `TParams` descendent which creates TSQLDBParams (885) descendents when a new field is added to the field set, to provide storage for engine-specific field data.

See also: TSQLDBParam (884)

39.23 TSQLHandle

39.23.1 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 (886) component to communicate with the TSQL-Connection (871) descendent.

This object must not be used directly.

See also: TSQLQuery (886), TSQLCursor (883)

39.24 TSQLQuery

39.24.1 Description

`TSQLQuery` exposes the properties and some methods introduced in TCustomSQLQuery (862). It encapsulates a single SQL statement: it implements all the necessary #fcl.db.TDataset (409) 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 (864) method.

Typically, the TSQLQuery.Database (894) property must be set once, the TSQLQuery.Transaction (894) property as well. Then the TSQLQuery.SQL (894) property can be set. Depending on the kind of SQL statement, the Open (428) 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 (915), TSQLConnection (871), TCustomSQLQuery.ExecSQL (864), TSQLQuery.SQL (894)

39.24.2 Property overview

Page	Properties	Access	Description
890	Active		
890	AfterCancel		
890	AfterClose		
890	AfterDelete		
891	AfterEdit		
891	AfterInsert		
891	AfterOpen		
891	AfterPost		
891	AfterRefresh		Event triggered after refresh.
891	AfterScroll		
890	AutoCalcFields		
891	BeforeCancel		
892	BeforeClose		
892	BeforeDelete		
892	BeforeEdit		
892	BeforeInsert		
892	BeforeOpen		
892	BeforePost		
892	BeforeRefresh		Event triggered before refresh.
893	BeforeScroll		
894	Database		The `TSQLConnection` instance on which to execute SQL Statements.
900	DataSource		Source for parameter values for unbound parameters.
896	DeleteSQL		Statement to be used when deleting a new row in the database.
889	FieldDefs		List of field definitions.
890	Filter		
890	Filtered		
896	IndexDefs		List of local index Definitions.
895	InsertSQL		Statement to be used when inserting a new row in the database.
898	MacroChar		Macro delimiter character.
898	MacroCheck		Check for macros in the SQL statement.
898	Macros		Set of macros for this SQL statement.
889	MaxIndexesCount		Maximum allowed number of indexes.
893	OnCalcFields		
893	OnDeleteError		
893	OnEditError		
893	OnFilterRecord		
893	OnNewRecord		
893	OnPostError		
897	Options		Options controlling the behaviour of the dataset.
898	ParamCheck		Should the SQL statement be checked for parameters.
897	Params		Parameters detected in the SQL statement.
899	ParseSQL		Should the SQL statement be parsed or not.
894	ReadOnly		
896	RefreshSQL		Refresh query to re-fetch field values after a DB update.
889	SchemaType		Schema type.
900	Sequence		Sequence to use for auto-generating values using a sequence.
901	ServerFilter		Append server-side filter to SQL statement.
901	ServerFiltered		Should server-side filter be applied.
901	ServerIndexDefs		List of indexes on the primary table of the query.
894	SQL		The SQL statement to execute.
889	StatementType		SQL statement type.
894	Transaction		Transaction in which to execute SQL statements.
899	UpdateMode		How to create update SQL statements.
895	UpdateSQL		Statement to be used when updating an existing row

39.24.3 TSQLQuery.SchemaType

Synopsis: Schema type.

Declaration: `Property SchemaType :`

Visibility: public

Access:

Description: `SchemaType` is the schema type set by TCustomSQLQuery.SetSchemaInfo (865). It determines what kind of schema information will be returned by the `TSQLQuery` instance.

See also: TCustomSQLQuery.SetSchemaInfo (865), RetrievingSchemaInformation (848)

39.24.4 TSQLQuery.StatementType

Synopsis: SQL statement type.

Declaration: `Property StatementType :`

Visibility: public

Access:

Description: `StatementType` is determined during the Prepare (864) call when ParseSQL (899) is set to `True`. It gives an indication of the type of SQL statement that is being executed.

See also: TSQLQuery.SQL (894), TSQLQuery.ParseSQL (899), TSQLQuery.Params (897)

39.24.5 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.

39.24.6 TSQLQuery.FieldDefs

Synopsis: List of field definitions.

Declaration: `Property FieldDefs :`

Visibility: published

Access:

39.24.7 TSQLQuery.Active

Declaration: `Property Active :`

Visibility: published

Access:

39.24.8 TSQLQuery.AutoCalcFields

Declaration: `Property AutoCalcFields :`

Visibility: published

Access:

39.24.9 TSQLQuery.Filter

Declaration: `Property Filter :`

Visibility: published

Access:

39.24.10 TSQLQuery.Filtered

Declaration: `Property Filtered :`

Visibility: published

Access:

39.24.11 TSQLQuery.AfterCancel

Declaration: `Property AfterCancel :`

Visibility: published

Access:

39.24.12 TSQLQuery.AfterClose

Declaration: `Property AfterClose :`

Visibility: published

Access:

39.24.13 TSQLQuery.AfterDelete

Declaration: `Property AfterDelete :`

Visibility: published

Access:

39.24.14 TSQLQuery.AfterEdit

Declaration: `Property AfterEdit :`

Visibility: published

Access:

39.24.15 TSQLQuery.AfterInsert

Declaration: `Property AfterInsert :`

Visibility: published

Access:

39.24.16 TSQLQuery.AfterOpen

Declaration: `Property AfterOpen :`

Visibility: published

Access:

39.24.17 TSQLQuery.AfterPost

Declaration: `Property AfterPost :`

Visibility: published

Access:

39.24.18 TSQLQuery.AfterRefresh

Synopsis: Event triggered after refresh.

Declaration: `Property AfterRefresh :`

Visibility: published

Access:

39.24.19 TSQLQuery.AfterScroll

Declaration: `Property AfterScroll :`

Visibility: published

Access:

39.24.20 TSQLQuery.BeforeCancel

Declaration: `Property BeforeCancel :`

Visibility: published

Access:

39.24.21 TSQLQuery.BeforeClose

Declaration: `Property BeforeClose :`

Visibility: published

Access:

39.24.22 TSQLQuery.BeforeDelete

Declaration: `Property BeforeDelete :`

Visibility: published

Access:

39.24.23 TSQLQuery.BeforeEdit

Declaration: `Property BeforeEdit :`

Visibility: published

Access:

39.24.24 TSQLQuery.BeforeInsert

Declaration: `Property BeforeInsert :`

Visibility: published

Access:

39.24.25 TSQLQuery.BeforeOpen

Declaration: `Property BeforeOpen :`

Visibility: published

Access:

39.24.26 TSQLQuery.BeforePost

Declaration: `Property BeforePost :`

Visibility: published

Access:

39.24.27 TSQLQuery.BeforeRefresh

Synopsis: Event triggered before refresh.

Declaration: `Property BeforeRefresh :`

Visibility: published

Access:

39.24.28 TSQLQuery.BeforeScroll

Declaration: `Property BeforeScroll :`

Visibility: published

Access:

39.24.29 TSQLQuery.OnCalcFields

Declaration: `Property OnCalcFields :`

Visibility: published

Access:

39.24.30 TSQLQuery.OnDeleteError

Declaration: `Property OnDeleteError :`

Visibility: published

Access:

39.24.31 TSQLQuery.OnEditError

Declaration: `Property OnEditError :`

Visibility: published

Access:

39.24.32 TSQLQuery.OnFilterRecord

Declaration: `Property OnFilterRecord :`

Visibility: published

Access:

39.24.33 TSQLQuery.OnNewRecord

Declaration: `Property OnNewRecord :`

Visibility: published

Access:

39.24.34 TSQLQuery.OnPostError

Declaration: `Property OnPostError :`

Visibility: published

Access:

39.24.35 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 (871)) 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 (894) property must be set.

Multiple `TSQLQuery` instances can be connected to a database at the same time.

See also: TSQLQuery.Transaction (894), TSQLConnection (871), TSQLTransaction (915)

39.24.36 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 (915)) 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 (894), TSQLConnection (871), TSQLTransaction (915)

39.24.37 TSQLQuery.ReadOnly

Declaration: `Property ReadOnly :`

Visibility: published

Access:

39.24.38 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 (864) is called, or Open (428) is called. It should contain a valid SQL statement for the connection to which the TSQLQuery (886) 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 (864)

If ParseSQL (899) is `True`, the SQL statement will be parsed and the Params (897) property will be updated with the names of the parameters found in the SQL statement.

See also Using parameters.

See also: TSQLQuery.ParseSQL (899), TSQLQuery.Params (897), TCustomSQLQuery.ExecSQL (864), TDataset.Open (428)

39.24.39 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 (894), TSQLQuery.UpdateSQL (895), TSQLQuery.DeleteSQL (896), TSQLQuery.UpdateMode (899), UsingParams (849), UpdateSQLS (848)

39.24.40 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 (894), TSQLQuery.InsertSQL (895), TSQLQuery.DeleteSQL (896), TSQLQuery.UpdateMode (899), UsingParams (849), UpdateSQLS (848)

39.24.41 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 (894), TSQLQuery.UpdateSQL (895), TSQLQuery.DeleteSQL (896), TSQLQuery.UpdateMode (899), UsingParams (849), UpdateSQLS (848)

39.24.42 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 (844) 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 (897)

See also: TField.Providerflags (844), TSQLQuery.Options (897)

39.24.43 TSQLQuery.IndexDefs

Synopsis: List of local index Definitions.

Declaration: `Property IndexDefs :`

Visibility: published

Access:

Description: List of local index Definitions.

See also: TCustomBufDataset.IndexDefs (844)

39.24.44 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:

sqoKeepOpenOnCommitThe 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.

sqoAutoApplyUpdatesSetting this option will make `TSQLQuery` call `ApplyUpdates` after every `Post` or `Delete` operation.

sqoAutoCommitSetting this option will make `TSQLQuery` call commit after every `ApplyUpdates`

sqoCancelUpdatesOnRefreshSetting 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

sqoPreferRefreshIf 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 (896) property or a constructed refresh SQL statement.

See also: TCustomSQLQuery.ApplyUpdates (866), TCustomSQLQuery.Post (866), TCustomSQLQuery.Delete (866)

39.24.45 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 (899) is `True`. For each named parameter in the SQL (894) property, a named item will appear in the collection, and the collection will be used to retrieve values from.

When Open (428) or ExecSQL (864) is called, and the Datasource (900) property is not `Nil`, then for each parameter for which no value was explicitly set (its Bound (541) 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 (894), TSQLQuery.ParseSQL (899), TParam.Bound (541), UsingParams (849), UpdateSQLS (848)

39.24.46 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 (530) instances should be added to the Params (897) 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 (530), Params (897), ParamCheck (898)

39.24.47 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 (897) 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 (897)

39.24.48 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 (898): if set to `True`, it instructs the query component to check the SQL statement text for macros and add them to the Macros (898) collection.

See also: ParamCheck (898), Macros (898)

39.24.49 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.

39.24.50 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 (894) property and attempting to detect the statement type or updating the Params (897) or StatementType (889) 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 (894), TSQLQuery.Params (897)

39.24.51 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 (895) and DeleteSQL (896) statements are auto-generated.

upWhereAllUse all old field values.

upWhereChangedUse only old field values of modified fields.

upWhereKeyOnlyOnly use key fields in the where clause.

See also: TSQLQuery.UpdateSQL (895), TSQLQuery.InsertSQL (895)

39.24.52 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 (486) of the fields in the primary key with pfInKey (365).

The effect of this is that when UpdateMode (899) 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 (848).

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 (899), #fcl.bufdataset.TCustomBufDataset.Unidirectional (183), TField.ProviderFlags (486), pfInKey (365), UpdateSQLs (848)

39.24.53 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 (428) or ExecSQL (864) is called, and the `Datasource` property is not `Nil` then for each parameter for which no value was explicitly set (its Bound (541) 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 (897), ExecSQL (864), UsingParams (849), TParam.Bound (541)

39.24.54 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 (909) 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 (911)).

See also: TSQLSequence (909)

39.24.55 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 (894), when ServerFiltered (901) is set to `True`. if ServerFiltered (901) is set to `False`, `ServerFilter` is ignored.

If the dataset is active and ServerFiltered (901) is set to true, then changing this property will re-fetch the data from the server.

This property cannot be used when ParseSQL (899) 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 (899) is `False` will result in an exception.

See also: ServerFiltered (901)

39.24.56 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 (901). 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 (899) is `False` will result in an exception.

See also: ParseSQL (899), ServerFilter (901)

39.24.57 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 (899) is `True`. If a primary key is found, then the fields in it will be marked

See also: UsePrimaryKeyAsKey (899), Prepare (864)

39.25 TSQLScript

39.25.1 Description

`TSQLScript` is a component that can be used to execute many SQL statements using a TSQLQuery (886) component. The SQL statements are specified in a script TSQLScript.Script (906) separated by a terminator character (typically a semicolon (;)).

See also: TSQLTransaction (915), TSQLConnection (871), TCustomSQLQuery.ExecSQL (864), TSQLQuery.SQL (894)

39.25.2 Method overview

Page	Method	Description
902	Create	Create a new `TSQLScript` instance.
903	Destroy	Remove the `TSQLScript` instance from memory.
903	Execute	Execute the script.
903	ExecuteScript	Convenience function, simply calls `Execute`.

39.25.3 Property overview

Page	Properties	Access	Description
903	Aborted		True when the script was aborted.
905	AutoCommit		Automatically commit every statement.
907	CommentsinSQL		Should comments be passed to the SQL engine ?
904	DataBase	rw	Database on which to execute the script.
906	Defines		Defined macros.
906	Directives		List of directives.
905	DollarStrings		List of alternate string delimiter token sequences.
904	Line		Current line of execution in the script.
904	OnDirective	rw	Event handler if a directive is encountered.
909	OnException		Exception handling event.
906	Script		The script to execute.
907	Terminator		Terminator character.
904	Transaction	rw	Transaction to use in the script.
908	UseCommit		Control automatic handling of the `COMMIT` command.
908	UseDefines		Automatically handle pre-processor defines.
905	UseDollarString		Enable support for dollarstrings.
907	UseSetTerm		Should the SET TERM directive be recognized.

39.25.4 TSQLScript.Create

Synopsis: Create a new `TSQLScript` instance.

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

Visibility: public

Description: `Create` instantiates a TSQLQuery (886) instance which will be used to execute the queries, and then calls the inherited constructor.

See also: TSQLScript.Destroy (903)

39.25.5 TSQLScript.Destroy

Synopsis: Remove the `TSQLScript` instance from memory.

Declaration: `destructor Destroy; Override`

Visibility: public

Description: `Destroy` frees the TSQLQuery (886) instance that was created during the `Create` constructor from memory and then calls the inherited destructor.

See also: TSQLScript.Create (902)

39.25.6 TSQLScript.Execute

Synopsis: Execute the script.

Declaration: `procedure Execute; Override`

Visibility: public

Description: `Execute` will execute the statements specified in Script (906) 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 (909) event handler is set, it may stop the event handler.

Errors: Handle errors using TSQLScript.OnException (909).

See also: Script (906), TSQLScript.OnException (909)

39.25.7 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.

39.25.8 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.

39.25.9 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.

39.25.10 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 (871) descendent. All SQL statements in the Script (906) property will be executed on this database.

See also: TSQLConnection (871), TSQLScript.Transaction (904), TSQLScript.Script (906)

39.25.11 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 (916) method.

See also: TSQLTransaction (915), TSQLTransaction.CommitRetaining (916), TSQLScript.Database (904)

39.25.12 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 (906) 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 (906)

39.25.13 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 (904)

39.25.14 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 (905) 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 (905)

39.25.15 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 (905) 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 (905)

39.25.16 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 (904) event handler. The list should contain one word per line, no spaces allowed.

See also: OnDirective (904)

39.25.17 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 (908) 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 (908)

39.25.18 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 (907) property. Each of the statement will be executed on the database specified in Database (904). using the equivalent of the TCustomSQLQuery.ExecSQL (864) 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 (907) property. If that property is `False`, comments will be stripped prior to executing the SQL statements.

See also: TSQLScript.CommentsinSQL (907), TSQLScript.Terminator (907), TSQLScript.DataBase (904)

39.25.19 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 (904), Directives (906)

39.25.20 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 (906)

39.25.21 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 (907) 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 (907), TSQLSCript.Script (906), TSQLSCript.Directives (906)

39.25.22 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 (904) 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 (904), TSQLScript.Directives (906)

39.25.23 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 (906) 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 (906), TSQLScript.Defines (906)

39.25.24 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 (903)

39.26 TSQLSequence

39.26.1 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 (886) and its properties determine which field must be auto-generated, and at what moment this value must be generated.

See also: TSQLConnection.GetSequenceNames (875), TSQLConnection.GetNextValue (876)

39.26.2 Method overview

Page	Method	Description
910	Apply	Apply a new value to a field.
910	Assign	Assign one `TSQLSequence` to another.
909	Create	Create a new instance.
910	GetNextValue	Get a next value for the sequence.

39.26.3 Property overview

Page	Properties	Access	Description
911	ApplyEvent	rw	When to apply the new value.
910	FieldName	rw	Field to apply sequence to.
911	IncrementBy	rw	Value to increment sequence with.
911	SequenceName	rw	Sequence name to get values from.

39.26.4 TSQLSequence.Create

Synopsis: Create a new instance.

Declaration: `constructor Create(AQuery: TCustomSQLQuery)`

Visibility: public

Description: `Create` instantiates a new sequence. It requires a TSQLQuery (886) instance, which it needs to have access to a connection.

See also: TSQLQuery (886)

39.26.5 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 (910), TSQLSequence.SequenceName (911), TSQLSequence.IncrementBy (911)

39.26.6 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 (911) using TSQLSequence.GetNextValue (910) to the field TSQLSequence.FieldName (910) 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 (910), TSQLSequence.FieldName (910), TSQLSequence.SequenceName (911), TSQLSequence.IncrementBy (911)

39.26.7 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 (911) using TSQLSequence.IncrementBy (911)

Errors: If the dataset is not attached to a connected database, an exception will be raised.

See also: TSQLSequence.FieldName (910), TSQLSequence.SequenceName (911), TSQLSequence.IncrementBy (911), TSQLSequence.Apply (910), TSQLConnection.GetNextValue (876)

39.26.8 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 (910) is called. It must be a valid fieldname of the dataset that owns the `TSQLSequence` instance.

See also: Apply (910), SequenceName (911), IncrementBy (911)

39.26.9 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 (910) is called.

See also: TSQLSequence.Apply (910), TSQLSequence.GetNextValue (910), TSQLSequence.IncrementBy (911)

39.26.10 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 (911) when TSQLSequence.GetNextValue (910) is called.

See also: TSQLSequence.Apply (910), TSQLSequence.GetNextValue (910), TSQLSequence.SequenceName (911)

39.26.11 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 (425) or Append (416) is called) or when a newly inserted record is saved (when Post (428) is called).

See also: TSQLSequenceApplyEvent (857)

39.27 TSQLStatement

39.27.1 Description

`TSQLStatement` is a descendent of TCustomSQLStatement (867) which simply publishes the protected properties of that component.

See also: TCustomSQLStatement (867)

39.27.2 Property overview

Page	Properties	Access	Description
912	Database		Database instance to execute statement on.
912	DataSource		Datasource to copy parameter values from.
913	MacroCheck		Check for macros in the SQL statement.
913	Macros		Set of macros for this SQL statement.
912	ParamCheck		Should SQL be checked for parameters.
913	Params		List of parameters.
914	ParseSQL		Parse the SQL statement.
914	SQL		The SQL statement to execute.
914	Transaction		The transaction in which the SQL statement should be executed.

39.27.3 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 (871) descendent. It must be set, together with Transaction (914) in order to be able to call Prepare (869) or Execute (869).

See also: Transaction (914), Prepare (869), Execute (869)

39.27.4 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 (449) instance. When Execute (869) 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 (449), Execute (869), #fcl.db.TParam.Bound (541)

39.27.5 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 (530) instances should be added to the Params (913) 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 (530), TSQLStatement.Params (913), TSQLQuery.ParamCheck (898)

39.27.6 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 (914) statement (in the form `:ParamName`). The collection is filled automatically if the ParamCheck (912) property is `True`.

See also: SQL (914), ParamCheck (912), ParseSQL (914)

39.27.7 TSQLStatement.MacroCheck

Synopsis: Check for macros in the SQL statement.

Declaration: `Property MacroCheck :`

Visibility: published

Access:

Description: `MacroCheck` is the macro equivalent of ParamCheck (912): if set to `True`, it instructs the query component to check the SQL statement text for macros and add them to the Macros (913) collection.

See also: ParamCheck (912), Macros (913)

39.27.8 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 (913) 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 (913)

39.27.9 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 (914) 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 (914), ParamCheck (912)

39.27.10 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 (869) is called.

If ParamCheck (912) 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 (913) 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 (870) function may be wrong in such a case.

See also: ParseSQL (914), CheckParams (911), Params (913), Prepare (869), RowsAffected (870)

39.27.11 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 (912) property. This must be set before Prepare (869) is called.

See also: Database (912), Prepare (869), TSQLTransaction (915)

39.28 TSQLTransaction

39.28.1 Description

`TSQLTransaction` represents the transaction in which one or more TSQLQuery (886) 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 (871) used in an application.

See also: TSQLQuery (886), TSQLConnection (871)

39.28.2 Method overview

Page	Method	Description
916	Commit	Commit the transaction, end transaction context.
916	CommitRetaining	Commit the transaction, retain transaction context.
915	Create	Create a new transaction.
915	Destroy	Destroy transaction component.
917	EndTransaction	End the transaction.
916	Rollback	Roll back all changes made in the current transaction.
917	RollbackRetaining	Roll back changes made in the transaction, keep transaction context.
917	StartTransaction	Start a new transaction.

39.28.3 Property overview

Page	Properties	Access	Description
918	Action	rw	Currently unused in SQLDB.
918	Database		Database for which this component is handling connections.
918	Handle	r	Low-level transaction handle.
919	Options	rw	Transaction options.
919	Params	rw	Transaction parameters.
918	SQLConnection	rw	Database as TSQLConnection.

39.28.4 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.

39.28.5 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.

39.28.6 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 (917) method. To commit data while retaining an active transaction, execute CommitRetaining (916) instead.

Errors: Executing `Commit` when no transaction is active will result in an exception. A transaction must be started by calling StartTransaction (917). If the database backend reports an error, an exception is raised as well.

See also: StartTransaction (917), CommitRetaining (916), Rollback (916), RollbackRetaining (917)

39.28.7 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 (916) instead.

Errors: Executing `CommitRetaining` when no transaction is active will result in an exception. A transaction must be started by calling StartTransaction (917). If the database backend reports an error, an exception is raised as well.

See also: StartTransaction (917), Retaining (916), Rollback (916), RollbackRetaining (917)

39.28.8 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 (917) 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 (917). If the database backend reports an error, an exception is raised as well.

See also: StartTransaction (917), CommitRetaining (916), Commit (916), RollbackRetaining (917)

39.28.9 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 (916) 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 (917). If the database backend reports an error, an exception is raised as well.

See also: StartTransaction (917), Commit (916), Rollback (916), CommitRetaining (916)

39.28.10 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 (916) or can be undone with a Rollback (916) 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 (917), Commit (916), Rollback (916), CommitRetaining (916), EndTransaction (917)

39.28.11 TSQLTransaction.EndTransaction

Synopsis: End the transaction.

Declaration: `procedure EndTransaction;  Override`

Visibility: public

Description: `EndTransaction` is equivalent to RollBack (916).

See also: RollBack (916)

39.28.12 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 (871) descendent.

39.28.13 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 (871)

39.28.14 TSQLTransaction.Action

Synopsis: Currently unused in SQLDB.

Declaration: `Property Action : TCommitRollbackAction`

Visibility: published

Access: Read,Write

Description: `Action` is currently unused in SQLDB.

39.28.15 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 (871) instance this transaction is handling transactions in. All datasets connected to this transaction component must have the same value for their Database (894) property.

See also: TSQLQuery.Database (894), TSQLConnection (871)

39.28.16 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 (871)

39.28.17 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.

stoUseImplicitUse 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).

stoExplicitStartWhen set, whenever an SQL statement is executed, the transaction must have been started explicitly. Default behaviour is that the TSQLStatement (911) or TSQLQuery (886) start the transaction as needed.

See also: TSQLStatement (911), TSQLQuery (886)

Chapter 40

Reference for unit 'SQLTypes'

40.1 Used units

Table 40.1: Used units by unit 'SQLTypes'

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

40.2 Constants, types and variables

40.2.1 Types

```
TDBEventType = (detCustom,detPrepare,detExecute,detFetch,detCommit
  ,
                detRollBack,detParamValue,detActualSQL)
```

Table 40.2: 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 (871) through the OnLog (879) event. event.

```
TDBEventTypes = Set of TDBEventType
```

TDBEventTypes is a set of TDBEventType (920) values, which is used to filter the set of event messages that should be sent. The TSQLConnection.LogEvents (880) 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 40.3: 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 (865) 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 40.4: 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 (886) component.

40.3 TSqlObjectIdenfier

40.3.1 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 (920), #fcl.sqldb.TSQLConnection.GetObjectNames (874)

40.3.2 Method overview

Page	Method	Description
922	Create	Create a new instance of a `TSqlObjectIdenfier` class.
923	FullName	Return the full name of the object.

40.3.3 Property overview

Page	Properties	Access	Description
923	ObjectName	rw	Name of the object in the database.
923	SchemaName	rw	Schema name.

40.3.4 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 (923) and TSqlObjectIdenfier.SchemaName (923) properties.

See also: TSqlObjectIdenfier.ObjectName (923), TSqlObjectIdenfier.SchemaName (923)

40.3.5 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 (923), TSqlObjectIdenfier.ObjectName (923)

40.3.6 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 (920), #fcl.sqldb.TSQLConnection.GetObjectNames (874), TSqlObjectIdenfier.ObjectName (923)

40.3.7 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 (923) property in order to create a unique name for the object.

See also: TSqlObjectIdenfierList (920), #fcl.sqldb.TSQLConnection.GetObjectNames (874), TSqlObjectIdenfier.SchemaName (923)

40.4 TSqlObjectIdentifierList

40.4.1 Method overview

Page	Method	Description
924	AddIdentifier	

40.4.2 Property overview

Page	Properties	Access	Description
924	Identifiers	rw	

40.4.3 TSqlObjectIdentifierList.AddIdentifier

Declaration:
```
function AddIdentifier : TSqlObjectIdenfier;  Overload
function AddIdentifier(const AObjectName: string;
                       const ASchemaName: string) : TSqlObjectIdenfier
                       ;  Overload
```

Visibility: public

40.4.4 TSqlObjectIdentifierList.Identifiers

Declaration:
```
Property Identifiers[Index: Integer]: TSqlObjectIdenfier; default
```

Visibility: public

Access: Read,Write

Chapter 41

Reference for unit 'streamcoll'

41.1 Used units

Table 41.1: Used units by unit 'streamcoll'

Name	Page
Classes	??
System	??
sysutils	??

41.2 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 (928) 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.

41.3 Procedures and functions

41.3.1 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 (927) and returns the read value. The value cannot be read and written across systems that have different endian values.

See also: ColReadDateTime (926), ColWriteBoolean (927), ColReadString (927), ColReadInteger (926), ColReadFloat (926), ColReadCurrency (926)

41.3.2 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 (927) and returns the read value. The value cannot be read and written across systems that have different endian values.

See also: ColReadDateTime (926), ColReadBoolean (925), ColReadString (927), ColReadInteger (926), ColReadFloat (926), ColWriteCurrency (927)

41.3.3 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 (927) and returns the read value. The value cannot be read and written across systems that have different endian values.

See also: ColWriteDateTime (927), ColReadBoolean (925), ColReadString (927), ColReadInteger (926), ColReadFloat (926), ColReadCurrency (926)

41.3.4 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 (928) and returns the read value. The value cannot be read and written across systems that have different endian values.

See also: ColReadDateTime (926), ColReadBoolean (925), ColReadString (927), ColReadInteger (926), ColWriteFloat (928), ColReadCurrency (926)

41.3.5 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 (928) and returns the read value. The value cannot be read and written across systems that have different endian values.

See also: ColReadDateTime (926), ColReadBoolean (925), ColReadString (927), ColWriteInteger (928), ColReadFloat (926), ColReadCurrency (926)

41.3.6 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 (928) and returns the read value. The value cannot be read and written across systems that have different endian values.

See also: ColReadDateTime (926), ColReadBoolean (925), ColWriteString (928), ColReadInteger (926), ColReadFloat (926), ColReadCurrency (926)

41.3.7 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 (925), ColWriteString (928), ColWriteInteger (928), ColWriteCurrency (927), ColWriteDateTime (927), ColWriteFloat (928)

41.3.8 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 (927), ColWriteString (928), ColWriteInteger (928), ColWriteDateTime (927), ColWriteFloat (928), ColReadCurrency (926)

41.3.9 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 (926), ColWriteBoolean (927), ColWriteString (928), ColWriteInteger (928), ColWriteFloat (928), ColWriteCurrency (927)

41.3.10 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 (927), ColWriteBoolean (927), ColWriteString (928), ColWriteInteger (928), ColReadFloat (926), ColWriteCurrency (927)

41.3.11 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 (927), ColWriteString (928), ColReadInteger (926), ColWriteCurrency (927), ColWriteDateTime (927)

41.3.12 ColWriteString

Synopsis: Write a string value to the stream.

Declaration: `procedure ColWriteString(S: TStream; AValue: string)`

Visibility: default

Description: `ColWriteString` writes the string value `AValue` to the stream `S`.

See also: ColWriteBoolean (927), ColReadString (927), ColWriteInteger (928), ColWriteCurrency (927), ColWriteDateTime (927), ColWriteFloat (928)

41.4 EStreamColl

41.4.1 Description

Exception raised when an error occurs when streaming the collection.

41.5 TStreamCollection

41.5.1 Description

`TStreamCollection` is a TCollection (**??**) descendent which implements 2 calls LoadFromStream (929) and SaveToStream (929) which load and save the contents of the collection to a stream.

The collection items must be descendents of the TStreamCollectionItem (930) class for the streaming to work correctly.

Note that the stream must be used to load collections of the same type.

See also: TStreamCollectionItem (930)

41.5.2 Method overview

Page	Method	Description
929	LoadFromStream	Load the collection from a stream.
929	SaveToStream	Load the collection from the stream.

41.5.3 Property overview

Page	Properties	Access	Description
929	Streaming	r	Indicates whether the collection is currently being written to stream.

41.5.4 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 (929). 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 (929)

41.5.5 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 (929). 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 (929)

41.5.6 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 (929), TStreamCollection.SaveToStream (929)

41.6 TStreamCollectionItem

41.6.1 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 (928) to save or load the item from the stream.

See also: TStreamCollection (928)

Chapter 42

Reference for unit 'streamex'

42.1 Used units

Table 42.1: Used units by unit 'streamex'

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

42.2 Overview

streamex implements some extensions to be used together with streams from the classes unit.

42.3 Constants, types and variables

42.3.1 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`.

42.4 TBidirBinaryObjectReader

42.4.1 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 (932), TDelphiReader (933)

42.4.2 Property overview

Page	Properties	Access	Description
932	Position	rw	Position in the stream.

42.4.3 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 (933) class.

See also: TDelphiReader (933)

42.5 TBidirBinaryObjectWriter

42.5.1 Description

`TBidirBinaryObjectReader` is a class descendent from TBinaryObjectWriter (**??**), which implements the necessary support for BiDi data.

See also: TBidirBinaryObjectWriter (932), TDelphiWriter (934)

42.5.2 Property overview

Page	Properties	Access	Description
932	Position	rw	Position in the stream.

42.5.3 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 (934) class.

See also: TDelphiWriter (934)

42.6 TDelphiReader

42.6.1 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 (932) driver class.

See also: TDelphiWriter (934), TBidirBinaryObjectReader (932)

42.6.2 Method overview

Page	Method	Description
933	GetDriver	Return the driver class as a TBidirBinaryObjectReader (932) class.
933	Read	Read data from stream.
933	ReadStr	Overrides the standard `ReadStr` method.

42.6.3 Property overview

Page	Properties	Access	Description
934	Position	rw	Position in the stream.

42.6.4 TDelphiReader.GetDriver

Synopsis: Return the driver class as a TBidirBinaryObjectReader (932) class.

Declaration:
```
function GetDriver : TBidirBinaryObjectReader
```

Visibility: public

Description: `GetDriver` simply returns the used driver and typecasts it as TBidirBinaryObjectReader (932) class.

See also: TBidirBinaryObjectReader (932)

42.6.5 TDelphiReader.ReadStr

Synopsis: Overrides the standard `ReadStr` method.

Declaration:
```
function ReadStr : string
```

Visibility: public

Description: `ReadStr` makes sure the TBidirBinaryObjectReader (932) methods are used, to store additional information about the stream position when reading the strings.

See also: TBidirBinaryObjectReader (932)

42.6.6 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 (932) class when reading.

See also: TBidirBinaryObjectReader (932), TDelphiReader.Position (934)

42.6.7 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 (933)

42.7 TDelphiWriter

42.7.1 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 (932) driver class.

See also: TDelphiReader (933), TBidirBinaryObjectWriter (932)

42.7.2 Method overview

Page	Method	Description
935	FlushBuffer	Flushes the stream buffer.
934	GetDriver	Return the driver class as a TBidirBinaryObjectWriter (932) class.
935	Write	Write raw data to the stream.
935	WriteStr	Write a string to the stream.
935	WriteValue	Write value type.

42.7.3 Property overview

Page	Properties	Access	Description
935	Position	rw	Position in the stream.

42.7.4 TDelphiWriter.GetDriver

Synopsis: Return the driver class as a TBidirBinaryObjectWriter (932) class.

Declaration: `function GetDriver : TBidirBinaryObjectWriter`

Visibility: public

Description: `GetDriver` simply returns the used driver and typecasts it as TBidirBinaryObjectWriter (932) class.

See also: TBidirBinaryObjectWriter (932)

42.7.5 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.

42.7.6 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.

42.7.7 TDelphiWriter.WriteStr

Synopsis: Write a string to the stream.

Declaration: `procedure WriteStr(const Value: string)`

Visibility: public

Description: `WriteStr` writes a string to the stream, forcing the use of the TBidirBinaryObjectWriter (932) class methods, which update the position of the stream.

See also: TBidirBinaryObjectWriter (932)

42.7.8 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 (932) methods, which update the position of the stream.

See also: TBidirBinaryObjectWriter (932)

42.7.9 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 (932) instance used when streaming.

See also: TBidirBinaryObjectWriter (932)

42.8 TFileReader

42.8.1 Description

`TFileReader` is a `TTextReader` descendent that takes a file on disk as the source of text data.

See also: TStreamReader (942), TTextReader (950)

42.8.2 Method overview

Page	Method	Description
937	Close	Close the file.
936	Create	Create a new instance of `TFileReader` for a disk file.
936	Destroy	Remove the `TFileReader` instance from memory.
937	ReadLine	Read a line of text.
937	Reset	Reset the stream to its original position.

42.8.3 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 (931) will be used. If the `ABufferSize` argument is less than MIN_BUFFER_SIZE (931), 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 (943), TFileStream (**??**), TFileStream.Create (**??**)

42.8.4 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 (936)

42.8.5 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 (951) method.

See also: TTextReader.Reset (951)

42.8.6 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 (951)

See also: TTextReader.Close (951)

42.8.7 TFileReader.ReadLine

Synopsis: Read a line of text.

Declaration: `procedure ReadLine(out AString: string); 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 (951) method.

See also: Eof (952), TTextReader.ReadLine (951)

42.9 TStreamHelper

42.9.1 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 (**??**)

42.9.2 Method overview

Page	Method	Description
941	ReadDouble	Read a double-precision floating point value from the stream.
940	ReadDWordBE	Read a DWord from the stream, big endian.
938	ReadDWordLE	Read a DWord from the stream, little endian.
940	ReadQWordBE	Read a QWord from the stream, big endian.
938	ReadQWordLE	Read a QWord from the stream, little endian.
941	ReadSingle	Read a single-precision floating point value from the stream.
940	ReadWordBE	Read a Word from the stream, big endian.
938	ReadWordLE	Read a Word from the stream, little endian.
942	WriteDouble	Write a double-precision floating point value to the stream.
941	WriteDWordBE	Write a DWord value, big endian.
939	WriteDWordLE	Write a DWord value, little endian.
941	WriteQWordBE	Write a QWord value, big endian.
939	WriteQWordLE	Write a QWord value, little endian.
942	WriteSingle	Write a single-precision floating point value to the stream.
940	WriteWordBE	Write a word value, big endian.
939	WriteWordLE	Write a word value, little endian.

42.9.3 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 (938), TStreamHelper.ReadQWordLE (938), TStreamHelper.WriteWordLE (939)

42.9.4 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 (938), TStreamHelper.ReadQWordLE (938), TStreamHelper.WriteDWordLE (939)

42.9.5 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 (938), TStreamHelper.ReadDWordLE (938), TStreamHelper.WriteQWordLE (939)

42.9.6 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 (938), TStreamHelper.WriteDWordLE (939), TStreamHelper.WriteQWordLE (939)

42.9.7 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 (938), TStreamHelper.WriteWordLE (939), TStreamHelper.WriteQWordLE (939)

42.9.8 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 (938), TStreamHelper.WriteDWordLE (939), TStreamHelper.WriteWordLE (939)

42.9.9 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 (940), TStreamHelper.ReadQWordBE (940), TStreamHelper.WriteWordBE (940)

42.9.10 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 (940), TStreamHelper.ReadQWordBE (940), TStreamHelper.WriteDWordBE (941)

42.9.11 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 (940), TStreamHelper.ReadDWordBE (940), TStreamHelper.WriteQWordBE (941)

42.9.12 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 (940), TStreamHelper.WriteDWordBE (941), TStreamHelper.WriteQWordBE (941)

42.9.13 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 (940), TStreamHelper.WriteWordBE (940), TStreamHelper.WriteQWordBE (941)

42.9.14 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 (940), TStreamHelper.WriteDWordBE (941), TStreamHelper.WriteWordBE (940)

42.9.15 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 (941), TStreamHelper.WriteSingle (942)

42.9.16 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 (941), TStreamHelper.WriteDouble (942)

42.9.17 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 (941), TStreamHelper.WriteDouble (942)

42.9.18 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 (941), TStreamHelper.WriteSingle (942)

42.10 TStreamReader

42.10.1 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 (950)

42.10.2 Method overview

Page	Method	Description
943	Close	Close and possibly free the stream.
943	Create	Create a new instance of `TStreamReader` from a stream.
943	Destroy	Destroy the `TStreamReader` instance.
944	ReadLine	Read a line of text.
943	Reset	Reset the stream to its original position.

42.10.3 Property overview

Page	Properties	Access	Description
944	BaseStream	r	The stream with the text data.
944	OwnsStream	rw	Should the stream be freed on close.

42.10.4 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
```

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 (931) will be used. If the `ABufferSize` argument is less than MIN_BUFFER_SIZE (931), 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 (943), TStream (**??**)

42.10.5 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 (944)) if `OwnsStream` is `True`.

See also: TStreamReader.OwnsStream (944), TStreamReader.Create (943), TStreamReader.BaseStream (944)

42.10.6 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 (951) method.

Errors: If the source stream (TStreamReader.BaseStream (944)) is not seekable, then this method may raise an exception.

See also: TStreamReader.BaseStream (944)

42.10.7 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 (944), TStreamReader.OwnsStream (944)

42.10.8 TStreamReader.ReadLine

Synopsis: Read a line of text.

Declaration: `procedure ReadLine(out AString: string); 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 (951) method.

See also: Eof (952), TTextReader.ReadLine (951)

42.10.9 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 (943) call.

Manipulating the stream between calls to TStreamReader.ReadLine (944) is not allowed, it will lead to wrong data being read from the stream.

See also: TStreamReader.Create (943), TStreamReader.OwnsStream (944)

42.10.10 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 (944) must be freed when TStreamReader.Close (943) is called or when the `TStreamReader` instance is destroyed.

See also: TStreamReader.BaseStream (944), TStreamReader.Close (943)

42.11 TStreamWriter

42.11.1 Method overview

Page	Method	Description
945	Close	
945	Create	
945	Destroy	
945	Flush	
946	OwnStream	
946	Write	
946	WriteLine	

42.11.2 Property overview

Page	Properties	Access	Description
946	AutoFlush	rw	
947	BaseStream	r	
947	Encoding	r	
947	NewLine	rw	

42.11.3 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

42.11.4 TStreamWriter.Destroy

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

Visibility: public

42.11.5 TStreamWriter.Close

Declaration:
```
procedure Close;  Override
```

Visibility: public

42.11.6 TStreamWriter.Flush

Declaration:
```
procedure Flush;  Override
```

Visibility: public

42.11.7 TStreamWriter.OwnStream

Declaration: `procedure OwnStream`

Visibility: public

42.11.8 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

42.11.9 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

42.11.10 TStreamWriter.AutoFlush

Declaration: `Property AutoFlush : Boolean`

Visibility: public

Access: Read,Write

42.11.11 TStreamWriter.NewLine

Declaration: `Property NewLine : string`

Visibility: public

Access: Read,Write

42.11.12 TStreamWriter.Encoding

Declaration: `Property Encoding : TEncoding`

Visibility: public

Access: Read

42.11.13 TStreamWriter.BaseStream

Declaration: `Property BaseStream : TStream`

Visibility: public

Access: Read

42.12 TStringReader

42.12.1 Description

`TStreamReader` is a `TTextReader` descendent that takes a single string as the source of text data.

See also: TTextReader (950)

42.12.2 Method overview

Page	Method	Description
948	Close	Close and possibly free the stream.
947	Create	Create a new instance of `TStreamReader` from a string.
948	Destroy	Free the `TStringReader` instance.
948	ReadLine	Read a line of text.
948	Reset	Reset the stream to its original position.

42.12.3 TStringReader.Create

Synopsis: Create a new instance of `TStreamReader` from a string.

Declaration:
```
constructor Create(const AString: string; ABufferSize: Integer)
                    ;  Virtual
constructor Create(const AString: string);  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 (931) will be used. If the `ABufferSize` argument is less than MIN_BUFFER_SIZE (931), 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 (943), TStream (**??**)

42.12.4 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 (947)

42.12.5 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 (951) method.

See also: TStreamReader.BaseStream (944)

42.12.6 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 (951) method.

See also: TTextReader.Close (951)

42.12.7 TStringReader.ReadLine

Synopsis: Read a line of text.

Declaration: `procedure ReadLine(out AString: string); 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 (951) method.

See also: Eof (952), TTextReader.ReadLine (951)

42.13 TStringWriter

42.13.1 Method overview

Page	Method	Description
949	Close	
949	Create	
949	Destroy	
949	Flush	
950	ToString	
949	Write	
950	WriteLine	

42.13.2 TStringWriter.Create

Declaration:
```
constructor Create;  Overload
constructor Create(aBuilder: TStringBuilder);  Overload
```

Visibility: public

42.13.3 TStringWriter.Destroy

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

Visibility: public

42.13.4 TStringWriter.Close

Declaration:
```
procedure Close;  Override
```

Visibility: public

42.13.5 TStringWriter.Flush

Declaration:
```
procedure Flush;  Override
```

Visibility: public

42.13.6 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

42.13.7 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

42.13.8 TStringWriter.ToString

Declaration:
```
function ToString : string;  Override
```

Visibility: public

42.14 TTextReader

42.14.1 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 (951) procedure. Several descendents of this class exist which implement the reader interface for several sources of text data: TStreamReader (942), TFileReader (936), TStringReader (947).

See also: TStreamReader (942), TFileReader (936), TStringReader (947), ReadLine (951)

42.14.2 Method overview

Page	Method	Description
951	Close	Close the text data stream.
951	Create	Instantiate a new instance.
951	ReadLine	Read a line of text.
951	Reset	Reset the reader to the start position.

42.14.3 Property overview

Page	Properties	Access	Description
952	Eof	r	Check whether the end of the text data is returned.

42.14.4 TTextReader.Create

Synopsis: Instantiate a new instance.

Declaration: `constructor Create; Virtual`

Visibility: public

Description: `Create` does nothing in `TTextReader`.

See also: ReadLine (951)

42.14.5 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 (951), TTextReader.ReadLine (951)

42.14.6 TTextReader.Close

Synopsis: Close the text data stream.

Declaration: `procedure Close; Virtual; Abstract`

Visibility: public

Description: `Close` closes the data stream. No ReadLine (951) call can be performed after a call to `Close`.

See also: TTextReader.Reset (951), TTextReader.ReadLine (951)

42.14.7 TTextReader.ReadLine

Synopsis: Read a line of text.

Declaration:
```
procedure ReadLine(out AString: string); Virtual; Abstract; Overload
function ReadLine : string; 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 (952)

42.14.8 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 (951)

42.15 TTextWriter

42.15.1 Method overview

Page	Method	Description
952	Close	
952	Flush	
952	Write	
953	WriteLine	

42.15.2 TTextWriter.Close

Declaration:
```
procedure Close;  Virtual;  Abstract
```

Visibility: public

42.15.3 TTextWriter.Flush

Declaration:
```
procedure Flush;  Virtual;  Abstract
```

Visibility: public

42.15.4 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

42.15.5 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

42.16 TWindowedStream

42.16.1 Description

`TWindowedStream` is a TStream (931) 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 (931), TWindowedStream.Create (954)

42.16.2 Method overview

Page	Method	Description
954	Create	Initialize a new instance of `TWindowedStream`.
954	Destroy	Destroy the `TWindowedStream` instance.
954	Read	Read data from the stream.
955	Seek	Reposition the stream.
954	Write	Read data to the stream.

42.16.3 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.

42.16.4 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 (954)

42.16.5 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 (954), TWindowedStream.Seek (955)

42.16.6 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 (954), TWindowedStream.Seek (955)

42.16.7 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 (954), TWindowedStream.Write (954)

Chapter 43

Reference for unit 'StreamIO'

43.1 Used units

Table 43.1: Used units by unit 'StreamIO'

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

43.2 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.

43.3 Procedures and functions

43.3.1 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 (957)

43.3.2 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 (956).

Errors: An invalid class reference will be returned if the file was not associated with a stream.

See also: AssignStream (956), TStream (**??**)

Chapter 44

Reference for unit 'syncobjs'

44.1 Used units

Table 44.1: Used units by unit 'syncobjs'

Name	Page
System	??
sysutils	??

44.2 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 (959) class is a wrapper around low-level critical section routines (semaphores or mutexes). The TEventObject (962) class can be used to send messages between threads (also known as conditional variables in POSIX threads).

44.3 Constants, types and variables

44.3.1 Constants

```
INFINITE = Cardinal(- 1)
```

Constant denoting an infinite timeout.

44.3.2 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 (962) class.

```
TEventHandle = Pointer
```

TEventHandle is an opaque type and should not be used in user code.

```
TWaitResult = (wrSignaled,wrTimeout,wrAbandoned,wrError)
```

Table 44.2: 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.

44.4 ELockException

44.4.1 Description

ELockException is provided for Delphi compatibility. It is not used in FPC.

See also: ESyncObjectException (959), ELockRecursionException (959)

44.5 ELockRecursionException

44.5.1 Description

ELockRecursionException is provided for Delphi compatibility. It is not used in FPC.

See also: ESyncObjectException (959), ELockException (959)

44.6 ESyncObjectException

44.6.1 Description

ESyncObjectException is used in the constructor of TEventObject (962) to indicate failure to create a basic event.

See also: TEventObject (962), ELockRecursionException (959), ELockException (959)

44.7 TCriticalSection

44.7.1 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 (960), Release (960)

44.7.2 Method overview

Page	Method	Description
960	Acquire	Enter the critical section.
961	Create	Create a new critical section.
962	Destroy	Destroy the criticalsection instance.
961	Enter	Alias for `Acquire`.
961	Leave	Alias for `Release`.
960	Release	Leave the critical section.
961	TryEnter	Try and obtain the critical section.

44.7.3 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 (960)

44.7.4 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 (960)

44.7.5 TCriticalSection.Enter

Synopsis: Alias for `Acquire`.

Declaration: `procedure Enter`

Visibility: public

Description: `Enter` just calls Acquire (960).

See also: Leave (961), Acquire (960)

44.7.6 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 (961)

44.7.7 TCriticalSection.Leave

Synopsis: Alias for `Release`.

Declaration: `procedure Leave`

Visibility: public

Description: `Leave` just calls Release (960)

See also: Release (960), Enter (961)

44.7.8 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 (962)

44.7.9 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 (961), Acquire (960)

44.8 TEventObject

44.8.1 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 (959)

44.8.2 Method overview

Page	Method	Description
962	Create	Create a new event object.
963	destroy	Clean up the event and release from memory.
963	ResetEvent	Reset the event.
963	SetEvent	Set the event.
963	WaitFor	Wait for the event to be set.

44.8.3 Property overview

Page	Properties	Access	Description
964	ManualReset	r	Should the event be reset manually.

44.8.4 TEventObject.Create

Synopsis: Create a new event object.

Declaration:
```
constructor Create(EventAttributes: PSecurityAttributes;
                   AManualReset: Boolean; InitialState: Boolean;
                   const Name: string; UseComWait: Boolean)
constructor Create(UseComWait: Boolean)
```

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 (964), ResetEvent (963)

44.8.5 TEventObject.destroy

Synopsis: Clean up the event and release from memory.

Declaration: `destructor destroy;  Override`

Visibility: public

Description: `Destroy` cleans up the low-level resources allocated for this event and releases the event instance from memory.

See also: Create (962)

44.8.6 TEventObject.ResetEvent

Synopsis: Reset the event.

Declaration: `procedure ResetEvent`

Visibility: public

Description: `ResetEvent` turns off the event. Any WaitFor (963) operation will suspend the calling thread.

See also: SetEvent (963), WaitFor (963)

44.8.7 TEventObject.SetEvent

Synopsis: Set the event.

Declaration: `procedure SetEvent`

Visibility: public

Description: `SetEvent` sets the event. If the ManualReset (964) is `True` any thread that was waiting for the event to be set (using WaitFor (963)) 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 (963), ManualReset (964)

44.8.8 TEventObject.WaitFor

Synopsis: Wait for the event to be set.

Declaration: `function WaitFor(Timeout: Cardinal) : TWaitResult`

Visibility: public

Description: `WaitFor` should be used in threads that should be notified when the event is set. When `WaitFor` is called, and the event is not set, the thread will be suspended. As soon as the event is set by some other thread (using SetEvent (963)) or the timeout period (`TimeOut`) has expired, the `WaitFor` function returns. The return value depends on the condition that caused the `WaitFor` function to return.

The calling thread will wait indefinitely when the constant `INFINITE` is specified for the `TimeOut` parameter.

See also: TEventObject.SetEvent (963)

44.8.9 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 (963) must be called manually after a thread was notified.

See also: ResetEvent (963)

44.9 THandleObject

44.9.1 Description

`THandleObject` is an abstract parent class for synchronization classes that need to store an operating system handle. It introduces a property Handle (965) 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 (965)

44.9.2 Method overview

Page	Method	Description
965	destroy	Free the instance.

44.9.3 Property overview

Page	Properties	Access	Description
965	Handle	r	Handle for this object.
965	LastError	r	Last operating system error.

44.9.4 THandleObject.destroy

Synopsis: Free the instance.

Declaration: `destructor destroy;  Override`

Visibility: public

Description: `Destroy` does nothing in the Free Pascal implementation of `THandleObject`.

44.9.5 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.

44.9.6 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 (965).

See also: Handle (965)

44.10 TSimpleEvent

44.10.1 Description

`TSimpleEvent` is a simple descendent of the TEventObject (962) class. It creates an event with no name, which must be reset manually, and which is initially not set.

See also: TEventObject (962), TSimpleEvent.Create (966)

44.10.2 Method overview

Page	Method	Description
966	Create	Creates a new `TSimpleEvent` instance.

44.10.3 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 (962) with `Nil` for the security attributes, an empty name, `AManualReset` set to `True`, and `InitialState` to `False`.

See also: TEventObject.Create (962)

44.11 TSynchroObject

44.11.1 Description

`TSynchroObject` is an abstract synchronization resource object. It implements 2 virtual methods Acquire (966) which can be used to acquire the resource, and Release (966) to release the resource.

See also: Acquire (966), Release (966)

44.11.2 Method overview

Page	Method	Description
966	Acquire	Acquire synchronization resource.
966	Release	Release previously acquired synchronization resource.

44.11.3 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 (966)

44.11.4 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 (966) call.

See also: Acquire (966)

Chapter 45

Reference for unit 'URIParser'

45.1 Used units

Table 45.1: Used units by unit 'URIParser'

Name	Page
System	??

45.2 Overview

The URIParser unit contains a basic type (TURI (969)) and some routines for the parsing (ParseURI (968)) and construction (EncodeURI (967)) 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..

45.3 Constants, types and variables

45.3.1 Types

45.4 Procedures and functions

45.4.1 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 (968)

45.4.2 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 (969)

45.4.3 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 (968), URIToFileName (969)

45.4.4 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 (967)

45.4.5 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`

45.4.6 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 (968), FilenameToURI (968)

45.5 TURI

```
TURI = record
  Protocol : string;
  Username : string;
  Password
  : string;
  Host : string;
  Port : Word;
  Path : string;
  Document
  : string;
  Params : string;
  Bookmark : string;
  HasAuthority
  : Boolean;
end
```

`TURI` is the basic record that can be filled by the ParseURI (968) call. It contains the contents of a URI, parsed out in it's various pieces.

Chapter 46

Reference for unit 'Zipper'

46.1 Used units

Table 46.1: Used units by unit 'Zipper'

Name	Page
BaseUnix	??
Classes	??
System	??
sysutils	??
ZStream	1009

46.2 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.

46.3 Constants, types and variables

46.3.1 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
```

46.3.2 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 (976)

```
FreeListArray = Array[FIRSTENTRY..TABLESIZE] of Word
```

Helper type in decoding the zip file.

```
FreeListPtr = ^FreeListArray
```

Pointer to FreeListArray (974)

```
TablePtr = ^CodeArray
```

Pointer to CodeArray (974)

```
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: string
  )
                                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.

46.4 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.

46.5 CodeRec

```
CodeRec = packed record
  Child : SmallInt;
  Sibling : SmallInt;
  Suffix : Byte;
end
```

Small LZW compression helper type.

46.6 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.

46.7 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.

46.8 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.

46.9 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.

46.10 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

46.11 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
```

46.12 EZipError

46.12.1 Description

Exception raised for errors in TZipper and TUnZipper.

46.13 TCompressor

46.13.1 Description

This object compresses a stream into a compressed zip stream.

46.13.2 Method overview

Page	Method	Description
979	Compress	Compresses input stream to output stream.
979	Create	Creates a TCompressor (978) object.
979	Terminate	Halts the compressor by setting the Terminated property to True.
979	ZipBitFlag	Current bit.
979	ZipID	Identifier for type of compression.
979	ZipVersionReqd	ZIP version required in the method.

46.13.3 Property overview

Page	Properties	Access	Description
980	BufferSize	r	Size of the buffer used for compression.
980	Crc32Val	rw	Running CRC32 value.
980	OnPercent	rw	Threshold percentage which triggers an OnProgress update.
980	OnProgress	rw	Event handler signalled to indicate the completion percentage for the compressor.
980	Terminated	r	Set to True when the Terminate method is called.

46.13.4 TCompressor.Create

Synopsis: Creates a TCompressor (978) object.

Declaration:
```
constructor Create(AInFile: TStream; AOutFile: TStream;
                   ABufSize: LongWord);  Virtual
```

Visibility: public

46.13.5 TCompressor.Compress

Synopsis: Compresses input stream to output stream.

Declaration:
```
procedure Compress;  Virtual;  Abstract
```

Visibility: public

46.13.6 TCompressor.ZipID

Synopsis: Identifier for type of compression.

Declaration:
```
class function ZipID : Word;  Virtual;  Abstract
```

Visibility: public

46.13.7 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.

46.13.8 TCompressor.ZipBitFlag

Synopsis: Current bit.

Declaration:
```
function ZipBitFlag : Word;  Virtual;  Abstract
```

Visibility: public

Description: Abstract virtual function. Must be implemented in a descendent class.

46.13.9 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.

46.13.10 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 (979), TShrinker.Compress (988), TDeflater.Compress (983)

46.13.11 TCompressor.OnPercent

Synopsis: Threshold percentage which triggers an OnProgress update.

Declaration: `Property OnPercent : Integer`

Visibility: public

Access: Read,Write

46.13.12 TCompressor.OnProgress

Synopsis: Event handler signalled to indicate the completion percentage for the compressor.

Declaration: `Property OnProgress : TProgressEvent`

Visibility: public

Access: Read,Write

46.13.13 TCompressor.Crc32Val

Synopsis: Running CRC32 value.

Declaration: `Property Crc32Val : LongWord`

Visibility: public

Access: Read,Write

Description: Running CRC32 value used when writing zip header.

46.13.14 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.

46.14 TDeCompressor

46.14.1 Description

This object decompresses a compressed zip stream.

46.14.2 Method overview

Page	Method	Description
981	Create	Creates decompressor object.
981	DeCompress	Decompress zip stream.
981	Terminate	Halts decompression and sets Terminated to True.
982	ZipID	Identifier for type of compression.

46.14.3 Property overview

Page	Properties	Access	Description
982	BufferSize	r	Size of buffer used in decompression.
982	Crc32Val	rw	Running CRC32 value used for verifying zip file integrity.
982	OnPercent	rw	Percentage of decompression completion.
982	OnProgress	rw	Event handler for OnProgress procedure.
982	OnProgressEx	rw	Event handler signalled to indicate progress using processed and total byte counts.
983	Terminated	r	Set to True when the Terminate method is called.

46.14.4 TDeCompressor.Create

Synopsis: Creates decompressor object.

Declaration:
```
constructor Create(AInFile: TStream; AOutFile: TStream;
                   ABufSize: LongWord);  Virtual
```

Visibility: public

46.14.5 TDeCompressor.DeCompress

Synopsis: Decompress zip stream.

Declaration:
```
procedure DeCompress;  Virtual;  Abstract
```

Visibility: public

46.14.6 TDeCompressor.Terminate

Synopsis: Halts decompression and sets Terminated to True.

Declaration:
```
procedure Terminate
```

Visibility: public

Description: Halts decompression and sets Terminated to True.

46.14.7 TDeCompressor.ZipID

Synopsis: Identifier for type of compression.

Declaration: `class function ZipID : Word; Virtual; Abstract`

Visibility: public

46.14.8 TDeCompressor.BufferSize

Synopsis: Size of buffer used in decompression.

Declaration: `Property BufferSize : LongWord`

Visibility: public

Access: Read

46.14.9 TDeCompressor.OnPercent

Synopsis: Percentage of decompression completion.

Declaration: `Property OnPercent : Integer`

Visibility: public

Access: Read,Write

46.14.10 TDeCompressor.OnProgress

Synopsis: Event handler for OnProgress procedure.

Declaration: `Property OnProgress : TProgressEvent`

Visibility: public

Access: Read,Write

46.14.11 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.

46.14.12 TDeCompressor.Crc32Val

Synopsis: Running CRC32 value used for verifying zip file integrity.

Declaration: `Property Crc32Val : LongWord`

Visibility: public

Access: Read,Write

46.14.13 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.

46.15 TDeflater

46.15.1 Description

Child of TCompressor (978) that implements the Deflate compression method.

46.15.2 Method overview

Page	Method	Description
983	Compress	Performs compression using the Deflate algorithm.
983	Create	Constructor for the class instance.
984	ZipBitFlag	Bitness flag.
984	ZipID	Zip algorithm ID.
984	ZipVersionReqd	Required version.

46.15.3 Property overview

Page	Properties	Access	Description
984	CompressionLevel	rw	Indicates the compression level applied in the Compress method.

46.15.4 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 (984), TCompressor.Create (979)

46.15.5 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.

46.15.6 TDeflater.ZipID

Synopsis: Zip algorithm ID.

Declaration: `class function ZipID : Word;  Override`

Visibility: public

Description: Zip algorithm ID.

46.15.7 TDeflater.ZipVersionReqd

Synopsis: Required version.

Declaration: `class function ZipVersionReqd : Word;  Override`

Visibility: public

Description: Required version.

46.15.8 TDeflater.ZipBitFlag

Synopsis: Bitness flag.

Declaration: `function ZipBitFlag : Word;  Override`

Visibility: public

Description: Bitness flag.

46.15.9 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:

clNoneDo not use compression, just copy data.

clFastestUse the fast (but less) compression.

clDefaultUse the default compression. dd

clMaxUse the maximum compression.

See also: TDeflater.Compress (983), TCompressionLevel (1009)

46.16 TFullZipFileEntries

46.16.1 Description

Collection of TFullZipFileEntry items.

See also: TFullZipFileEntry (985)

46.16.2 Property overview

Page	Properties	Access	Description
985	FullEntries	rw	Array access to all entries.

46.16.3 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 (985)

46.17 TFullZipFileEntry

46.17.1 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.

46.17.2 Property overview

Page	Properties	Access	Description
986	BitFlags	r	General purpose bit flag from the Local Header in the .ZIP archive file.
986	CompressedSize	r	Size after applying the compression method and level.
986	CompressMethod	r	Compression method for the file.
986	CRC32	rw	32-bit CRC value for the file.

46.17.3 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.

46.17.4 TFullZipFileEntry.CompressMethod

Synopsis: Compression method for the file.

Declaration: `Property CompressMethod : Word`

Visibility: public

Access: Read

Description: Compression method for the file.

46.17.5 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.

46.17.6 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.

46.18 TInflater

46.18.1 Description

Child of TDeCompressor (981) that implements the Inflate decompression method.

46.18.2 Method overview

Page	Method	Description
987	Create	Constructor for the class instance.
987	DeCompress	Removes compression applied using the deflate algorithm.
987	ZipID	Zip algorithm ID.

46.18.3 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.

46.18.4 TInflater.DeCompress

Synopsis: Removes compression applied using the deflate algorithm.

Declaration:
```
procedure DeCompress;  Override
```

Visibility: public

Description: Removes compression applied using the deflate algorithm.

46.18.5 TInflater.ZipID

Synopsis: Zip algorithm ID.

Declaration:
```
class function ZipID : Word;  Override
```

Visibility: public

Description: Zip algorithm ID.

46.19 TShrinker

46.19.1 Description

TShrinker implements the LZW lossless data compression algorithm created by Abraham Lempel, Jacob Ziv, and Terry Welch also known as "shrink" compression.

46.19.2 Method overview

Page	Method	Description
988	Compress	Compresses input values using LZW (shrink) compression.
988	Create	Constructor for the class instance.
988	Destroy	Destructor for the class instance.
988	ZipBitFlag	Zip bitness flag.
988	ZipID	Return Zip algorithm ID.
988	ZipVersionReqd	Minimum zip algorithm required.

46.19.3 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.

46.19.4 TShrinker.Destroy

Synopsis: Destructor for the class instance.

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

Visibility: public

Description: Destructor for the class instance.

46.19.5 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.

46.19.6 TShrinker.ZipID

Synopsis: Return Zip algorithm ID.

Declaration:
```
class function ZipID : Word;  Override
```

Visibility: public

Description: Return Zip algorithm ID.

46.19.7 TShrinker.ZipVersionReqd

Synopsis: Minimum zip algorithm required.

Declaration:
```
class function ZipVersionReqd : Word;  Override
```

Visibility: public

Description: Minimum zip algorithm required.

46.19.8 TShrinker.ZipBitFlag

Synopsis: Zip bitness flag.

Declaration:
```
function ZipBitFlag : Word;  Override
```

Visibility: public

Description: Zip bitness flag.

46.20 TUnZipper

46.20.1 Description

Extracts and decompresses files and directories in a .ZIP archive file.

46.20.2 Method overview

Page	Method	Description
991	Clear	Removes all entries and files from object.
990	Create	Constructor for the class instance.
990	Destroy	Destructor for the class instance.
992	Examine	Opens zip file and reads the directory entries (list of zipped files).
992	Terminate	Sets the value in Terminated to True.
991	Unzip	Unzips the specified .ZIP archive file.
990	UnZipAllFiles	Unzips all files in a zip file, writing them to disk.
990	UnZipFile	Unzips a single file found in the specified .ZIP archive.
991	UnZipFiles	Unzips the specified files in a .ZIP archive file.

46.20.3 Property overview

Page	Properties	Access	Description
992	BufferSize	rw	Size of the buffer used to read and decompress entries in the .ZIP file.
995	Entries	r	Collection with TFullZipFileEntry instances for files and directories stored in the .ZIP archive.
994	FileComment	r	Comment stored in the .ZIP archive file.
994	FileName	rw	Path and file name for the .zip file to be unzipped / processed.
995	Files	r	Files in the zip file (deprecated).
996	Flat	rw	Extracts files to a single directory.
992	OnCloseInputStream	rw	Event handler signalled when the input stream for the .ZIP file is closed.
993	OnCreateStream	rw	Event handler signalled when an output stream is created.
993	OnDoneStream	rw	Event handler signalled when an output stream is closed.
994	OnEndFile	rw	Callback procedure that will be called after unzipping a file.
992	OnOpenInputStream	rw	Event handler signalled when the input stream for the .ZIP file is opened.
993	OnPercent	rw	Threshold percentage which triggers a progress notification.
993	OnProgress	rw	Progress event handler used when decompressing files.
993	OnProgressEx	rw	Extended progress event handler used when decompressing files.
994	OnStartFile	rw	Callback procedure that will be called before unzipping a file.
994	OutputPath	rw	Path where archive files will be unzipped.
996	Terminated	r	True if the Terminate method has been called.
995	UseUTF8	rw	Indicates that the UTF-8-encoded names are used when locating and unzipping entries in the archive.

46.20.4 TUnZipper.Create

Synopsis: Constructor for the class instance.

Declaration:
```
constructor Create
```

Visibility: public

Description: Constructor for the class instance.

46.20.5 TUnZipper.Destroy

Synopsis: Destructor for the class instance.

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

Visibility: public

Description: Destructor for the class instance.

46.20.6 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 (1001) 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.
```

46.20.7 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.

46.20.8 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.

46.20.9 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: string)
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.

46.20.10 TUnZipper.Clear

Synopsis: Removes all entries and files from object.

Declaration:
```
procedure Clear
```

Visibility: public

Description: Removes all entries and files from object.

46.20.11 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).

46.20.12 TUnZipper.Terminate

Synopsis: Sets the value in Terminated to True.

Declaration: `procedure Terminate`

Visibility: public

Description: Sets the value in Terminated to True.

46.20.13 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.

46.20.14 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.

46.20.15 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.

46.20.16 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.

46.20.17 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.

46.20.18 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.

46.20.19 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.

46.20.20 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.

46.20.21 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.

46.20.22 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.

46.20.23 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.

46.20.24 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.

46.20.25 TUnZipper.FileComment

Synopsis: Comment stored in the .ZIP archive file.

Declaration: `Property FileComment : string`

Visibility: public

Access: Read

Description: Comment stored in the .ZIP archive file.

46.20.26 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.

46.20.27 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 (992), TUnZipper.Clear (991), TUnZipper.UnZipAllFiles (990), TUnZipper.UnZipFiles (991), TUnZipper.UnZipFile (990), TUnZipper.UnZip (991), TFullZipFileEntries (985), TFullZipFileEntry (985)

46.20.28 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 (995), TUnZipper.UnZipAllFiles (990), TUnZipper.UnZipFiles (991), TUnZipper.UnZipFile (990), TZipFileEntries (996), TZipFileEntry (997)

46.20.29 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.

46.20.30 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.

46.21 TZipFileEntries

46.21.1 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 (997), TZipFileEntry (997), TZipper.Entries (1007), TUnZipper.Entries (995), TFullZipFileEntries (985)

46.21.2 Method overview

Page	Method	Description
997	AddFileEntries	Adds TZipFileEntry instances in the collection for the file names in List.
996	AddFileEntry	Adds file to zip directory.

46.21.3 Property overview

Page	Properties	Access	Description
997	Entries	rw	Entries (files) in the zip archive.

46.21.4 TZipFileEntries.AddFileEntry

Synopsis: Adds file to zip directory.

Declaration:
```
function AddFileEntry(const ADiskFileName: string) : TZipFileEntry
function AddFileEntry(const ADiskFileName: string;
                      const AArchiveFileName: string) : TZipFileEntry
function AddFileEntry(const AStream: TStream;
                      const AArchiveFileName: string) : 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.

46.21.5 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 (996)

46.21.6 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 (997), TCollection.Items (**??**)

46.22 TZipFileEntry

46.22.1 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.

46.22.2 Method overview

Page	Method	Description
999	Assign	Copies property values from the specified persistent object.
998	Create	Constructor for the class instance.
999	IsDirectory	True if the entry is a directory on the local file system.
999	IsLink	True if the directory is a symbolic link on the local file system.

46.22.3 Property overview

Page	Properties	Access	Description
1000	ArchiveFileName	rw	Name of the file or directory in the .ZIP archive.
1001	Attributes	rw	File attributes for the file or directory.
1001	CompressionLevel	rw	Compression level applied to the content stored in the .ZIP archive.
1001	DateTime	rw	Timestamp for the file or directory in the .ZIP archive.
1000	DiskFileName	rw	Name of the file or directory on the local file system.
1001	OS	rw	Indication of operating system/file system.
1000	Size	rw	Size of the compressed content for the file or directory.
999	Stream	rw	Stream with the content for the entry.
1000	UTF8ArchiveFileName	rw	Archive filename as UTF8 string.
1000	UTF8DiskFileName	rw	Name of the file or directory on the local file system using UTF-8 encoding.

46.22.4 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:

DateTimeSets to the current date and time for the local computer.

OSSet to OS_UNIX for UNIX-like environments, or OS_VFAT for all others.

AttributesSet to 0 (no attributes).

CompressionLevelSet to clDefault.

Create calls the inherited constructor prior to exiting from the method.

46.22.5 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.

46.22.6 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.

46.22.7 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.

46.22.8 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.

46.22.9 TZipFileEntry.ArchiveFileName

Synopsis: Name of the file or directory in the .ZIP archive.

Declaration: `Property ArchiveFileName : string`

Visibility: published

Access: Read,Write

Description: Name of the file or directory in the .ZIP archive.

46.22.10 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.

46.22.11 TZipFileEntry.DiskFileName

Synopsis: Name of the file or directory on the local file system.

Declaration: `Property DiskFileName : string`

Visibility: published

Access: Read,Write

Description: Name of the file or directory on the local file system.

46.22.12 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.

46.22.13 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.

46.22.14 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.

46.22.15 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.

46.22.16 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.

46.22.17 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.

46.23 TZipper

46.23.1 Description

Creates a .ZIP archive file.

46.23.2 Method overview

Page	Method	Description
1005	Clear	Removes all values in the Entries and Files properties.
1002	Create	Constructor for the class instance.
1002	Destroy	Destructor for the class instance.
1003	SaveToFile	Saves the archive to a file with a new name.
1003	SaveToStream	Save the archive to a stream.
1005	Terminate	Halts an assigned compressor in the class instance, and sets Terminated to True.
1004	Zip	Convenience method used to create a .zip file with the given name containing the specified file(s).
1003	ZipAllFiles	Zips all files in object and writes zip to disk.
1004	ZipFile	Zip one file to a zip file.
1004	ZipFiles	Zip multiple files into an archive.

46.23.3 Property overview

Page	Properties	Access	Description
1005	BufferSize	rw	Buffer size used when reading and processing files.
1007	Entries	rw	Collection with the TZipFileEntry instances in the .ZIP archive.
1006	FileComment	rw	Comment stored in the .ZIP archive file.
1006	FileName	rw	Name of the .ZIP archive file where the compressed files and directories are stored.
1007	Files	r	Provides access to the list of files and directories in the archive.
1007	InMemSize	rw	Total memory used for the compressed content in the .ZIP file.
1006	OnEndFile	rw	Event handler signalled when compression for a file has been completed.
1005	OnPercent	rw	Threshold percentage which triggers progress notifications when processing files.
1006	OnProgress	rw	Event handler signalled to show a percent complete progress notifications.
1006	OnStartFile	rw	Event handler signalled when compression for a file is started.
1007	Terminated	r	True if the Terminate method has been called.
1008	UseLanguageEncoding	rw	Use language encoding.

46.23.4 TZipper.Create

Synopsis: Constructor for the class instance.

Declaration: `constructor Create`

Visibility: public

Description: Constructor for the class instance.

46.23.5 TZipper.Destroy

Synopsis: Destructor for the class instance.

Declaration: `destructor Destroy;  Override`

Visibility: public

Description: Destructor for the class instance.

46.23.6 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 (1001) 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.
```

46.23.7 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.

46.23.8 TZipper.SaveToStream

Synopsis: Save the archive to a stream.

Declaration: `procedure SaveToStream(AStream: TStream)`

Visibility: public

Description: Save the archive to a stream.

46.23.9 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 (1004)

46.23.10 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 (1004)

46.23.11 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 (991)

46.23.12 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.

46.23.13 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.

46.23.14 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.

46.23.15 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.

46.23.16 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.

46.23.17 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.

46.23.18 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.

46.23.19 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.

46.23.20 TZipper.FileComment

Synopsis: Comment stored in the .ZIP archive file.

Declaration: `Property FileComment : string`

Visibility: public

Access: Read,Write

Description: Comment stored in the .ZIP archive file.

46.23.21 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 (1007), TZipperFileEntries.AddFileEntry (970), TZipperFileEntries.AddFileEntries (970)

46.23.22 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.

46.23.23 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.

46.23.24 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.

46.23.25 TZipper.UseLanguageEncoding

Synopsis: Use language encoding.

Declaration: `Property UseLanguageEncoding : Boolean`

Visibility: public

Access: Read,Write

Description: EFS/language encoding using UTF-8.

Chapter 47

Reference for unit 'ZStream'

47.1 Used units

Table 47.1: Used units by unit 'ZStream'

Name	Page
Classes	**??**
gzio	**??**
System	**??**
zbase	**??**

47.2 Overview

The ZStream unit implements a TStream (**??**) descendent (TCompressionStream (1010)) 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 (1013)) and decompresses it with the inflate algorithm.

47.3 Constants, types and variables

47.3.1 Types

```
Tcompressionlevel = (clnone,clfastest,cldefault,clmax)
```

Table 47.2: 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 47.3: Enumeration values for type Tgzopenmode

Value	Explanation
gzopenread	Open file for reading.
gzopenwrite	Open file for writing.

Open mode for gzip file.

47.4 Ecompressionerror

47.4.1 Description

`ECompressionError` is the exception class used by the TCompressionStream (1010) class.

47.5 Edecompressionerror

47.5.1 Description

`EDecompressionError` is the exception class used by the TDeCompressionStream (1013) class.

47.6 Egzfileerror

47.6.1 Description

`Egzfileerror` is the exception class used to report errors by the Tgzfilestream (1016) class.

See also: Tgzfilestream (1016)

47.7 Ezliberror

47.7.1 Description

Errors which occur in the zstream unit are signaled by raising an `EZLibError` exception descendent.

47.8 Tcompressionstream

47.8.1 Description

`TCompressionStream`

47.8.2 Method overview

Page	Method	Description
1011	create	Create a new instance of the compression stream.
1011	destroy	Flushes data to the output stream and destroys the compression stream.
1012	flush	Flush remaining data to the target stream.
1012	get_compressionrate	Get the current compression rate.
1011	write	Write data to the stream.

47.8.3 Property overview

Page	Properties	Access	Description
1012	OnProgress		Progress handler.

47.8.4 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 (1011)

47.8.5 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 (1011)

47.8.6 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 (1010) exception is raised.

See also: Write (1011), ECompressionError (1010)

47.8.7 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.

47.8.8 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.

47.8.9 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.

47.9 Tcustomzlibstream

47.9.1 Description

`TCustomZlibStream` serves as the ancestor class for the TCompressionStream (1010) and TDeCompressionStream (1013) classes.

It introduces support for a progress handler, and stores the input or output stream.

47.9.2 Method overview

Page	Method	Description
1013	create	Create a new instance of `TCustomZlibStream`.
1013	destroy	Clear up instance.

47.9.3 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 (1010), TDecompressionStream (1013)

47.9.4 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 (1013)

47.10 Tdecompressionstream

47.10.1 Description

`TDecompressionStream` performs the inverse operation of TCompressionStream (1010). 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 (1010) compression stream.

See also: TCompressionStream (1010)

47.10.2 Method overview

Page	Method	Description
1014	create	Creates a new instance of the `TDecompressionStream` stream.
1014	destroy	Destroys the `TDecompressionStream` instance.
1015	get_compressionrate	Get the current compression rate.
1014	read	Read data from the compressed stream.
1015	Seek	Move stream position to a certain location in the stream.

47.10.3 Property overview

Page	Properties	Access	Description
1015	OnProgress		Progress handler.

47.10.4 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 (1014)

47.10.5 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 (1014)

47.10.6 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 (1010) exception is raised.

See also: Write (1011)

47.10.7 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 (1010) exception is raised if the stream does not allow the requested seek operation.

See also: Read (1014)

47.10.8 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.

47.10.9 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.

47.11 TGZFileStream

47.11.1 Description

`TGZFileStream` can be used to read data from a gzip file, or to write data to a gzip file.

See also: TCompressionStream (1010), TDeCompressionStream (1013)

47.11.2 Method overview

Page	Method	Description
1016	create	Create a new instance of `TGZFileStream`.
1017	destroy	Removes `TGZFileStream` instance.
1016	read	Read data from the compressed file.
1017	seek	Set the position in the compressed stream.
1017	write	Write data to be compressed.

47.11.3 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 (1016) 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 (1017) method will be stored in the file in compressed (deflated) form.

Errors: If the file is not found, an EZlibError (1010) exception is raised.

See also: Destroy (1017), TGZOpenMode (1010)

47.11.4 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 (1016), Write (1017), Seek (1017)

47.11.5 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 (1010) exception is raised.

See also: Create (1016), Read (1016), Seek (1017)

47.11.6 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 (1015) for a list of possibilities.

Errors: In case an impossible combination is asked, an EZlibError (1010) exception is raised.

See also: TDecompressionStream.Seek (1015)

47.11.7 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 (1016)

47.12 TGZipCompressionStream

47.12.1 Method overview

Page	Method	Description
1017	Create	
1018	Destroy	
1018	Write	

47.12.2 TGZipCompressionStream.Create

Declaration:
```
constructor Create(ADest: TStream);  Overload
constructor Create(ALevel: Tcompressionlevel; ADest: TStream);  Overload
```

Visibility: public

47.12.3 TGZipCompressionStream.Destroy

Declaration: `destructor Destroy;  Override`

Visibility: public

47.12.4 TGZipCompressionStream.Write

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

Visibility: public

47.13 TGZipDecompressionStream

47.13.1 Method overview

Page	Method	Description
1018	Create	
1018	Destroy	
1018	Read	
1018	Seek	

47.13.2 TGZipDecompressionStream.Create

Declaration: `constructor Create(ASource: TStream)`

Visibility: public

47.13.3 TGZipDecompressionStream.Destroy

Declaration: `destructor Destroy;  Override`

Visibility: public

47.13.4 TGZipDecompressionStream.Read

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

Visibility: public

47.13.5 TGZipDecompressionStream.Seek

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

Visibility: public