## Reference for unit 'AVL_TreeNode'

2.1 Used units ................................................. 96
2.2 Overview .................................................. 96
2.3 Constants, types and variables ................................. 96
  2.3.1 Types .................................................. 96
  2.3.2 Variables .............................................. 97
2.4 TAVLTree .................................................. 97
  2.4.1 Description ............................................. 97
  2.4.2 Method overview ....................................... 98
  2.4.3 Property overview ..................................... 99
  2.4.4 TAVLTree.Create ...................................... 99
  2.4.5 TAVLTree.CreateObjectCompare ......................... 99
  2.4.6 TAVLTree.Destroy ..................................... 99
  2.4.7 TAVLTree.SetNodeManager ............................. 99
  2.4.8 TAVLTree.NewNode .................................... 100
  2.4.9 TAVLTree.DisposeNode ................................ 100
  2.4.10 TAVLTree.Add ........................................ 100
  2.4.11 TAVLTree.AddAscendingSequence ...................... 101
  2.4.12 TAVLTree.Delete ..................................... 101
  2.4.13 TAVLTree.Remove ..................................... 101
  2.4.14 TAVLTree.RemovePointer ............................. 102
  2.4.15 TAVLTree.MoveDataLeftMost ........................ 102
  2.4.16 TAVLTree.MoveDataRightMost ......................... 102
  2.4.17 TAVLTree.Clear ...................................... 102
  2.4.18 TAVLTree.FreeAndClear .............................. 103
  2.4.19 TAVLTree.FreeAndDelete ............................. 103
  2.4.20 TAVLTree.Equals ..................................... 103
  2.4.21 TAVLTree.IsEqual .................................... 103
  2.4.22 TAVLTree.Assign ..................................... 104
  2.4.23 TAVLTree.Compare .................................... 104
  2.4.24 TAVLTree.Find ....................................... 104
  2.4.25 TAVLTree.FindKey ................................... 104
  2.4.26 TAVLTree.FindNearestKey ........................... 105
  2.4.27 TAVLTree.FindSuccessor ............................. 105
2.4.28 TAVLTree.FindPredecessor ........................................ 105
2.4.29 TAVLTree.FindLowest ........................................... 106
2.4.30 TAVLTree.FindHighest ........................................... 106
2.4.31 TAVLTree.FindNearest ........................................... 106
2.4.32 TAVLTree.FindPointer ........................................... 106
2.4.33 TAVLTree.FindLeftMost ........................................ 107
2.4.34 TAVLTree.FindRightMost ....................................... 107
2.4.35 TAVLTree.FindLeftMostKey .................................... 107
2.4.36 TAVLTree.FindRightMostKey ................................... 107
2.4.37 TAVLTree.FindLeftMostSameKey ................................. 108
2.4.38 TAVLTree.FindRightMostSameKey ............................... 108
2.4.39 TAVLTree.GetEnumerator ......................................... 108
2.4.40 TAVLTree.GetEnumeratorHighToLow ................................ 108
2.4.41 TAVLTree.ConsistencyCheck .................................... 109
2.4.42 TAVLTree.WriteReportToStream ................................ 109
2.4.43 TAVLTree.NodeToReportStr ..................................... 109
2.4.44 TAVLTree.ReportAsString ....................................... 109
2.4.45 TAVLTree.OnCompare ........................................... 110
2.4.46 TAVLTree.OnObjectCompare .................................... 110
2.4.47 TAVLTree.NodeClass ............................................ 110
2.4.48 TAVLTree.Root .................................................. 110
2.4.49 TAVLTree.Count ................................................. 111

2.5 TAVLTreeNode .......................................................... 111
  2.5.1 Description ...................................................... 111
  2.5.2 Method overview ................................................. 111
  2.5.3 TAVLTreeNode.Successor ....................................... 111
  2.5.4 TAVLTreeNode.Predecessor .................................... 111
  2.5.5 TAVLTreeNode.Clear ............................................ 112
  2.5.6 TAVLTreeNode.TreeDepth ....................................... 112
  2.5.7 TAVLTreeNode.ConsistencyCheck ................................ 112
  2.5.8 TAVLTreeNode.GetCount ....................................... 112

2.6 TAVLTreeNodeEnumerator ........................................... 112
  2.6.1 Description ...................................................... 112
  2.6.2 Method overview ................................................. 113
  2.6.3 Property overview .............................................. 113
  2.6.4 TAVLTreeNodeEnumerator.Create ............................... 113
  2.6.5 TAVLTreeNodeEnumerator.GetEnumerator ....................... 113
  2.6.6 TAVLTreeNodeEnumerator.MoveNext ............................ 113
  2.6.7 TAVLTreeNodeEnumerator.Current ............................. 113
  2.6.8 TAVLTreeNodeEnumerator.LowToHigh .......................... 114
CONTENTS

3.7.2 Method overview ............................................. 122
3.7.3 TBase64EncodingStream.Destroy .......................... 122
3.7.4 TBase64EncodingStream.Flush .............................. 122
3.7.5 TBase64EncodingStream.Write .............................. 123
3.7.6 TBase64EncodingStream.Seek ............................... 123

4 Reference for unit 'BlowFish' ................................. 124
  4.1 Used units ...................................................... 124
  4.2 Overview ...................................................... 124
  4.3 Constants, types and variables ............................... 124
     4.3.1 Constants .................................................. 124
     4.3.2 Types ...................................................... 124
  4.4 EBlowFishError ................................................. 125
     4.4.1 Description ................................................. 125
  4.5 TBlowFish ..................................................... 125
     4.5.1 Description ................................................. 125
     4.5.2 Method overview ........................................... 125
     4.5.3 TBlowFish.Create ......................................... 125
     4.5.4 TBlowFish.Encrypt ......................................... 126
     4.5.5 TBlowFish.Decrypt ......................................... 126
  4.6 TBlowFishDecrytStream ....................................... 126
     4.6.1 Description ................................................. 126
     4.6.2 Method overview ........................................... 126
     4.6.3 TBlowFishDecrytStream.Create ............................ 126
     4.6.4 TBlowFishDecrytStream.Read .............................. 127
     4.6.5 TBlowFishDecrytStream.Seek .............................. 127
  4.7 TBlowFishEncryptStream ..................................... 127
     4.7.1 Description ................................................. 127
     4.7.2 Method overview ........................................... 128
     4.7.3 TBlowFishEncryptStream.Destroy ......................... 128
     4.7.4 TBlowFishEncryptStream.Write ............................ 128
     4.7.5 TBlowFishEncryptStream.Seek .............................. 128
     4.7.6 TBlowFishEncryptStream.Flush ................................ 129
  4.8 TBlowFishStream .............................................. 129
     4.8.1 Description ................................................. 129
     4.8.2 Method overview ........................................... 129
     4.8.3 Property overview ......................................... 129
     4.8.4 TBlowFishStream.Create ................................... 129
     4.8.5 TBlowFishStream.Destroy .................................. 130
     4.8.6 TBlowFishStream.BlowFish ............................... 130
5 Reference for unit 'BufDataset'

5.1 Used units .......................................................... 131
5.2 Overview .............................................................. 131
5.3 Constants, types and variables ................................. 131
  5.3.1 Types ............................................................ 131
5.4 Procedures and functions ....................................... 133
  5.4.1 RegisterDatapacketReader .................................. 133
5.5 TBlobBuffer .......................................................... 134
5.6 TBufBlobField ....................................................... 134
5.7 TBufBookmark ....................................................... 134
5.8 TBufRecLinkItem ................................................... 134
5.9 TDBCompareRec ..................................................... 134
5.10 TRecUpdateBuffer ................................................ 135
5.11 TArrayBufIndex .................................................... 135
  5.11.1 Description ..................................................... 135
  5.11.2 Method overview .............................................. 136
  5.11.3 TArrayBufIndex.Create ...................................... 136
  5.11.4 TArrayBufIndex.ScrollBackward ............................ 136
  5.11.5 TArrayBufIndex.ScrollForward .............................. 136
  5.11.6 TArrayBufIndex.GetCurrent ................................. 136
  5.11.7 TArrayBufIndex.ScrollFirst ............................... 137
  5.11.8 TArrayBufIndex.ScrollLast ............................... 137
  5.11.9 TArrayBufIndex.SetToFirstRecord ......................... 137
  5.11.10 TArrayBufIndex.SetToLastRecord ....................... 137
  5.11.11 TArrayBufIndex.StoreCurrentRecord ..................... 137
  5.11.12 TArrayBufIndex.RestoreCurrentRecord .................. 137
  5.11.13 TArrayBufIndex.CanScrollForward ...................... 138
  5.11.14 TArrayBufIndex.DoScrollForward ....................... 138
  5.11.15 TArrayBufIndex.StoreCurrentRecIntoBookmark .......... 138
  5.11.16 TArrayBufIndex.StoreSpareRecIntoBookmark ............ 138
  5.11.17 TArrayBufIndex.GotoBookmark ............................ 138
  5.11.18 TArrayBufIndex.InitialiseIndex ....................... 138
  5.11.19 TArrayBufIndex.InitialiseSpareRecord ................. 138
  5.11.20 TArrayBufIndex.ReleaseSpareRecord .................... 139
  5.11.21 TArrayBufIndex.BeginUpdate ............................ 139
  5.11.22 TArrayBufIndex.AddRecord .............................. 139
  5.11.23 TArrayBufIndex.InsertRecordBeforeCurrentRecord .... 139
  5.11.24 TArrayBufIndex.RemoveRecordFromIndex ............... 139
  5.11.25 TArrayBufIndex.EndUpdate ............................. 139
5.12 TBufBlobStream .................................................. 140
5.18.2 Method overview ........................................... 178
5.18.3 TFpcBinaryDatapacketReader.Create .................... 178
5.18.4 TFpcBinaryDatapacketReader.LoadFieldDefs ............... 179
5.18.5 TFpcBinaryDatapacketReader.StoreFieldDefs ............... 179
5.18.6 TFpcBinaryDatapacketReader.InitLoadRecords ............. 179
5.18.7 TFpcBinaryDatapacketReader.GetCurrentRecord .......... 179
5.18.8 TFpcBinaryDatapacketReader.GetRecordRowState ........... 179
5.18.9 TFpcBinaryDatapacketReader.RestoreRecord ............... 179
5.18.10 TFpcBinaryDatapacketReader.GotoNextRecord ............. 179
5.18.11 TFpcBinaryDatapacketReader.StoreRecord ................. 180
5.18.12 TFpcBinaryDatapacketReader.FinalizeStoreRecords ....... 180
5.18.13 TFpcBinaryDatapacketReader.RecognizeStream ............ 180
5.19 TUniDirectionalBufIndex .................................. 180
5.19.1 Description ............................................... 180
5.19.2 Method overview ........................................... 181
5.19.3 TUniDirectionalBufIndex.ScrollBackward .................. 181
5.19.4 TUniDirectionalBufIndex.ScrollForward ................... 181
5.19.5 TUniDirectionalBufIndex.GetCurrent ...................... 181
5.19.6 TUniDirectionalBufIndex.ScrollFirst ..................... 181
5.19.7 TUniDirectionalBufIndex.ScrollLast ...................... 182
5.19.8 TUniDirectionalBufIndex.SetToFirstRecord ............... 182
5.19.9 TUniDirectionalBufIndex.SetToLastRecord ............... 182
5.19.10 TUniDirectionalBufIndex.StoreCurrentRecord .......... 182
5.19.11 TUniDirectionalBufIndex.RestoreCurrentRecord ......... 182
5.19.12 TUniDirectionalBufIndex.CanScrollForward ............. 182
5.19.13 TUniDirectionalBufIndex.DoScrollForward .............. 183
5.19.14 TUniDirectionalBufIndex.StoreCurrentRecIntoBookmark .. 183
5.19.15 TUniDirectionalBufIndex.StoreSpareRecIntoBookmark .... 183
5.19.16 TUniDirectionalBufIndex.GotoBookmark .................. 183
5.19.17 TUniDirectionalBufIndex.InitialiseIndex ............... 183
5.19.18 TUniDirectionalBufIndex.InitialiseSpareRecord ........ 183
5.19.19 TUniDirectionalBufIndex.ReleaseSpareRecord .......... 183
5.19.20 TUniDirectionalBufIndex.BeginUpdate ................... 184
5.19.21 TUniDirectionalBufIndex.AddRecord ..................... 184
5.19.22 TUniDirectionalBufIndex.InsertRecordBeforeCurrentRecord .... 184
5.19.23 TUniDirectionalBufIndex.RemoveRecordFromIndex ....... 184
5.19.24 TUniDirectionalBufIndex.OrderCurrentRecord .......... 184
5.19.25 TUniDirectionalBufIndex.EndUpdate ..................... 184

6 Reference for unit 'bufstream' ................................. 185
6.1 Used units ................................................................. 185
6.2 Overview ................................................................. 185
6.3 Constants, types and variables ........................................ 185
   6.3.1 Constants .......................................................... 185
6.4 TBufferedFileStream .................................................... 186
   6.4.1 Description ....................................................... 186
   6.4.2 Method overview ................................................ 186
   6.4.3 TBufferedFileStream.Create .................................... 186
   6.4.4 TBufferedFileStream.Destroy ................................... 187
   6.4.5 TBufferedFileStream.Seek ....................................... 187
   6.4.6 TBufferedFileStream.Read ................................. 188
   6.4.7 TBufferedFileStream.Write ................................... 188
   6.4.8 TBufferedFileStream.Flush ................................... 189
   6.4.9 TBufferedFileStream.InitializeCache ...................... 189
6.5 TBufStream .......................................................... 189
   6.5.1 Description ....................................................... 189
   6.5.2 Method overview ................................................ 190
   6.5.3 Property overview ............................................ 190
   6.5.4 TBufStream.Create ............................................. 190
   6.5.5 TBufStream.Destroy ............................................ 190
   6.5.6 TBufStream.Buffer ............................................ 190
   6.5.7 TBufStream.Capacity .......................................... 191
   6.5.8 TBufStream.BufferPos ........................................ 191
   6.5.9 TBufStream.BufferSize ....................................... 191
6.6 TReadBufStream ....................................................... 191
   6.6.1 Description ....................................................... 191
   6.6.2 Method overview ................................................ 192
   6.6.3 TReadBufStream.Seek ........................................ 192
   6.6.4 TReadBufStream.Read .......................................... 192
6.7 TWriteBufStream ...................................................... 192
   6.7.1 Description ....................................................... 192
   6.7.2 Method overview ................................................ 192
   6.7.3 TWriteBufStream.Destroy ..................................... 193
   6.7.4 TWriteBufStream.Seek ......................................... 193
   6.7.5 TWriteBufStream.Write ....................................... 193
7 Reference for unit 'CacheCls' ........................................ 194
   7.1 Used units .......................................................... 194
   7.2 Overview ........................................................... 194
   7.3 Constants, types and variables .................................. 194
CONTENTS

7.3.1 Resource strings ........................................... 194
7.3.2 Types ......................................................... 194
7.4 TCacheSlot ...................................................... 195
7.5 ECacheError ..................................................... 195
  7.5.1 Description ................................................. 195
7.6 TCache .......................................................... 195
  7.6.1 Description ................................................. 195
  7.6.2 Method overview ........................................... 196
  7.6.3 Property overview ....................................... 196
  7.6.4 TCache.Create ............................................. 196
  7.6.5 TCache.Destroy ............................................ 196
  7.6.6 TCache.Add ............................................... 196
  7.6.7 TCache.AddNew ............................................ 197
  7.6.8 TCache.FindSlot ......................................... 197
  7.6.9 TCache.IndexOf ......................................... 197
  7.6.10 TCache.Remove ......................................... 198
  7.6.11 TCache.Data ............................................. 198
  7.6.12 TCache.MRUSlot ......................................... 198
  7.6.13 TCache.LRUSlot ......................................... 199
  7.6.14 TCache.SlotCount ....................................... 199
  7.6.15 TCache.Slots ............................................ 199
  7.6.16 TCache.OnIsDataEqual ................................ 199
  7.6.17 TCache.OnFreeSlot ..................................... 200

8 Reference for unit 'Cntnrs' ..................................... 201
  8.1 Used units .................................................. 201
  8.2 Overview ................................................... 201
  8.3 Constants, types and variables ............................ 201
    8.3.1 Constants ................................................. 201
    8.3.2 Types ..................................................... 202
  8.4 Procedures and functions .................................. 205
    8.4.1 RSHash ................................................... 205
  8.5 TBucket ........................................................ 205
  8.6 TBucketItem .................................................. 205
  8.7 THashItem ..................................................... 206
  8.8 EDuplicate .................................................... 206
    8.8.1 Description ............................................... 206
  8.9 EKeyNotFound ................................................ 206
    8.9.1 Description ............................................... 206
  8.10 TBucketList .................................................. 206
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>8.10.1</td>
<td>Description</td>
<td>206</td>
</tr>
<tr>
<td>8.10.2</td>
<td>Method overview</td>
<td>206</td>
</tr>
<tr>
<td>8.10.3</td>
<td>TBucketList.Create</td>
<td>206</td>
</tr>
<tr>
<td>8.11</td>
<td>TClassList</td>
<td>207</td>
</tr>
<tr>
<td>8.11.1</td>
<td>Description</td>
<td>207</td>
</tr>
<tr>
<td>8.11.2</td>
<td>Method overview</td>
<td>207</td>
</tr>
<tr>
<td>8.11.3</td>
<td>Property overview</td>
<td>207</td>
</tr>
<tr>
<td>8.11.4</td>
<td>TClassList.Add</td>
<td>207</td>
</tr>
<tr>
<td>8.11.5</td>
<td>TClassList.Extract</td>
<td>208</td>
</tr>
<tr>
<td>8.11.6</td>
<td>TClassList.Remove</td>
<td>208</td>
</tr>
<tr>
<td>8.11.7</td>
<td>TClassList.IndexOf</td>
<td>208</td>
</tr>
<tr>
<td>8.11.8</td>
<td>TClassList.First</td>
<td>208</td>
</tr>
<tr>
<td>8.11.9</td>
<td>TClassList.Last</td>
<td>209</td>
</tr>
<tr>
<td>8.11.10</td>
<td>TClassList.Insert</td>
<td>209</td>
</tr>
<tr>
<td>8.11.11</td>
<td>TClassList.Items</td>
<td>209</td>
</tr>
<tr>
<td>8.12</td>
<td>TComponentList</td>
<td>209</td>
</tr>
<tr>
<td>8.12.1</td>
<td>Description</td>
<td>209</td>
</tr>
<tr>
<td>8.12.2</td>
<td>Method overview</td>
<td>210</td>
</tr>
<tr>
<td>8.12.3</td>
<td>Property overview</td>
<td>210</td>
</tr>
<tr>
<td>8.12.4</td>
<td>TComponentList.Destroy</td>
<td>210</td>
</tr>
<tr>
<td>8.12.5</td>
<td>TComponentList.Add</td>
<td>210</td>
</tr>
<tr>
<td>8.12.6</td>
<td>TComponentList.Extract</td>
<td>210</td>
</tr>
<tr>
<td>8.12.7</td>
<td>TComponentList.Remove</td>
<td>211</td>
</tr>
<tr>
<td>8.12.8</td>
<td>TComponentList.IndexOf</td>
<td>211</td>
</tr>
<tr>
<td>8.12.9</td>
<td>TComponentList.First</td>
<td>211</td>
</tr>
<tr>
<td>8.12.10</td>
<td>TComponentList.Last</td>
<td>212</td>
</tr>
<tr>
<td>8.12.11</td>
<td>TComponentList.Insert</td>
<td>212</td>
</tr>
<tr>
<td>8.12.12</td>
<td>TComponentList.Items</td>
<td>212</td>
</tr>
<tr>
<td>8.13</td>
<td>TCustomBucketList</td>
<td>212</td>
</tr>
<tr>
<td>8.13.1</td>
<td>Description</td>
<td>212</td>
</tr>
<tr>
<td>8.13.2</td>
<td>Method overview</td>
<td>213</td>
</tr>
<tr>
<td>8.13.3</td>
<td>Property overview</td>
<td>213</td>
</tr>
<tr>
<td>8.13.4</td>
<td>TCustomBucketList.Destroy</td>
<td>213</td>
</tr>
<tr>
<td>8.13.5</td>
<td>TCustomBucketList.Clear</td>
<td>213</td>
</tr>
<tr>
<td>8.13.6</td>
<td>TCustomBucketList.Add</td>
<td>214</td>
</tr>
<tr>
<td>8.13.7</td>
<td>TCustomBucketList.Assign</td>
<td>214</td>
</tr>
<tr>
<td>8.13.8</td>
<td>TCustomBucketList.Exists</td>
<td>214</td>
</tr>
<tr>
<td>8.13.9</td>
<td>TCustomBucketList.Find</td>
<td>214</td>
</tr>
<tr>
<td>8.13.10</td>
<td>TCustomBucketList.ForEach</td>
<td>215</td>
</tr>
<tr>
<td>8.13.11</td>
<td>TCustomBucketList.Remove</td>
<td>215</td>
</tr>
<tr>
<td>Section</td>
<td>Page</td>
<td></td>
</tr>
<tr>
<td>-----------------------</td>
<td>------</td>
<td></td>
</tr>
<tr>
<td>8.13.12 TCustomBucketList.Data</td>
<td>215</td>
<td></td>
</tr>
<tr>
<td>8.14 TFPCustomHashTable</td>
<td>215</td>
<td></td>
</tr>
<tr>
<td>8.14.1 Description</td>
<td>215</td>
<td></td>
</tr>
<tr>
<td>8.14.2 Method overview</td>
<td>216</td>
<td></td>
</tr>
<tr>
<td>8.14.3 Property overview</td>
<td>216</td>
<td></td>
</tr>
<tr>
<td>8.14.4 TFPCustomHashTable.Create</td>
<td>216</td>
<td></td>
</tr>
<tr>
<td>8.14.5 TFPCustomHashTable.CreateWith</td>
<td>217</td>
<td></td>
</tr>
<tr>
<td>8.14.6 TFPCustomHashTable.Destroy</td>
<td>217</td>
<td></td>
</tr>
<tr>
<td>8.14.7 TFPCustomHashTable.ChangeTableSize</td>
<td>217</td>
<td></td>
</tr>
<tr>
<td>8.14.8 TFPCustomHashTable.Clear</td>
<td>217</td>
<td></td>
</tr>
<tr>
<td>8.14.9 TFPCustomHashTable.Delete</td>
<td>218</td>
<td></td>
</tr>
<tr>
<td>8.14.10 TFPCustomHashTable.Find</td>
<td>218</td>
<td></td>
</tr>
<tr>
<td>8.14.11 TFPCustomHashTable.IsEmpty</td>
<td>218</td>
<td></td>
</tr>
<tr>
<td>8.14.12 TFPCustomHashTable.HashFunction</td>
<td>218</td>
<td></td>
</tr>
<tr>
<td>8.14.13 TFPCustomHashTable.Count</td>
<td>219</td>
<td></td>
</tr>
<tr>
<td>8.14.14 TFPCustomHashTable.HashTableSize</td>
<td>219</td>
<td></td>
</tr>
<tr>
<td>8.14.15 TFPCustomHashTable.HashTable</td>
<td>219</td>
<td></td>
</tr>
<tr>
<td>8.14.16 TFPCustomHashTable.VoidSlots</td>
<td>219</td>
<td></td>
</tr>
<tr>
<td>8.14.17 TFPCustomHashTable.LoadFactor</td>
<td>220</td>
<td></td>
</tr>
<tr>
<td>8.14.18 TFPCustomHashTable.AVGChainLen</td>
<td>220</td>
<td></td>
</tr>
<tr>
<td>8.14.19 TFPCustomHashTable.MaxChainLength</td>
<td>220</td>
<td></td>
</tr>
<tr>
<td>8.14.20 TFPCustomHashTable.NumberOfCollisions</td>
<td>220</td>
<td></td>
</tr>
<tr>
<td>8.14.21 TFPCustomHashTable.Density</td>
<td>221</td>
<td></td>
</tr>
<tr>
<td>8.15 TFPDataHashTable</td>
<td>221</td>
<td></td>
</tr>
<tr>
<td>8.15.1 Description</td>
<td>221</td>
<td></td>
</tr>
<tr>
<td>8.15.2 Method overview</td>
<td>221</td>
<td></td>
</tr>
<tr>
<td>8.15.3 Property overview</td>
<td>221</td>
<td></td>
</tr>
<tr>
<td>8.15.4 TFPDataHashTable.Iterate</td>
<td>221</td>
<td></td>
</tr>
<tr>
<td>8.15.5 TFPDataHashTable.Add</td>
<td>222</td>
<td></td>
</tr>
<tr>
<td>8.15.6 TFPDataHashTable.Items</td>
<td>222</td>
<td></td>
</tr>
<tr>
<td>8.16 TFPHashList</td>
<td>222</td>
<td></td>
</tr>
<tr>
<td>8.16.1 Description</td>
<td>222</td>
<td></td>
</tr>
<tr>
<td>8.16.2 Method overview</td>
<td>223</td>
<td></td>
</tr>
<tr>
<td>8.16.3 Property overview</td>
<td>223</td>
<td></td>
</tr>
<tr>
<td>8.16.4 TFPHashList.Create</td>
<td>223</td>
<td></td>
</tr>
<tr>
<td>8.16.5 TFPHashList.Destroy</td>
<td>223</td>
<td></td>
</tr>
<tr>
<td>8.16.6 TFPHashList.Add</td>
<td>224</td>
<td></td>
</tr>
<tr>
<td>8.16.7 TFPHashList.Clear</td>
<td>224</td>
<td></td>
</tr>
<tr>
<td>8.16.8 TFPHashList.NameOfIndex</td>
<td>224</td>
<td></td>
</tr>
<tr>
<td>8.16.9 TFPHashList.HashOfIndex</td>
<td>224</td>
<td></td>
</tr>
</tbody>
</table>
8.16.10 TFPHashList.GetNextCollision .............................. 225
8.16.11 TFPHashList.Delete ............................................. 225
8.16.12 TFPHashList.Error ................................................. 225
8.16.13 TFPHashList.Expand ............................................... 225
8.16.14 TFPHashList.Extract ............................................. 225
8.16.15 TFPHashList.IndexOf ........................................... 226
8.16.16 TFPHashList.Find ............................................... 226
8.16.17 TFPHashList.FindIndexOf ..................................... 226
8.16.18 TFPHashList.FindWithHash ................................... 226
8.16.19 TFPHashList.Rename ............................................ 227
8.16.20 TFPHashList.Remove ............................................ 227
8.16.21 TFPHashList.Pack ................................................. 227
8.16.22 TFPHashList.ShowStatistics ................................. 227
8.16.23 TFPHashList.ForEachCall ..................................... 228
8.16.24 TFPHashList.Capacity ......................................... 228
8.16.25 TFPHashList.Count ............................................. 228
8.16.26 TFPHashList.Items ............................................... 228
8.16.27 TFPHashList.List ................................................. 229
8.16.28 TFPHashList.Strs ................................................. 229
8.17 TFPHashObject ........................................................ 229
  8.17.1 Description ...................................................... 229
  8.17.2 Method overview ............................................... 229
  8.17.3 Property overview ............................................. 229
  8.17.4 TFPHashObject.CreateNotOwned .............................. 230
  8.17.5 TFPHashObject.Create ......................................... 230
  8.17.6 TFPHashObject.ChangeOwner .................................. 230
  8.17.7 TFPHashObject.ChangeOwnerAndName ......................... 230
  8.17.8 TFPHashObject.Rename ......................................... 231
  8.17.9 TFPHashObject.Name ............................................ 231
  8.17.10 TFPHashObject.Hash .......................................... 231
8.18 TFPHashObjectList .................................................. 232
  8.18.1 Method overview ............................................... 232
  8.18.2 Property overview ............................................. 232
  8.18.3 TFPHashObjectList.Create .................................. 232
  8.18.4 TFPHashObjectList.Destroy .................................. 232
  8.18.5 TFPHashObjectList.Clear ..................................... 233
  8.18.6 TFPHashObjectList.Add ....................................... 233
  8.18.7 TFPHashObjectList.NameOfIndex ............................ 233
  8.18.8 TFPHashObjectList.HashOfIndex ............................ 234
  8.18.9 TFPHashObjectList.GetNextCollision ....................... 234
8.18.10 TFPHashObjectList.Delete ........................................ 234
8.18.11 TFPHashObjectList.Expand ....................................... 234
8.18.12 TFPHashObjectList.Extract ....................................... 235
8.18.13 TFPHashObjectList.Remove ....................................... 235
8.18.14 TFPHashObjectList.IndexOf ...................................... 235
8.18.15 TFPHashObjectList.Find .......................................... 235
8.18.16 TFPHashObjectList.FindIndexOf .................................. 236
8.18.17 TFPHashObjectList.FindWithHash .................................. 236
8.18.18 TFPHashObjectList.Rename ....................................... 236
8.18.19 TFPHashObjectList.FindInstanceOf ................................ 236
8.18.20 TFPHashObjectList.Pack ......................................... 237
8.18.21 TFPHashObjectList.ShowStatistics ................................ 237
8.18.22 TFPHashObjectList.ForEachCall .................................. 237
8.18.23 TFPHashObjectList.Capacity ..................................... 237
8.18.24 TFPHashObjectList.Count ........................................ 238
8.18.25 TFPHashObjectList.OwnsObjects .................................. 238
8.18.26 TFPHashObjectList.Items ......................................... 238
8.18.27 TFPHashObjectList.List .......................................... 238

8.19 TFPObjectHashTable .................................................. 239
8.19.1 Description ......................................................... 239
8.19.2 Method overview .................................................... 239
8.19.3 Property overview .................................................. 239
8.19.4 TFPObjectHashTable.Create ....................................... 239
8.19.5 TFPObjectHashTable.CreateWith .................................... 239
8.19.6 TFPObjectHashTable.Iterate ....................................... 240
8.19.7 TFPObjectHashTable.Add ........................................... 240
8.19.8 TFPObjectHashTable.Items ........................................ 240
8.19.9 TFPObjectHashTable.OwnsObjects .................................. 240

8.20 TFPObjectList ......................................................... 241
8.20.1 Description .......................................................... 241
8.20.2 Method overview .................................................... 241
8.20.3 Property overview .................................................. 242
8.20.4 TFPObjectList.Create .............................................. 242
8.20.5 TFPObjectList.Destroy ............................................. 242
8.20.6 TFPObjectList.Clear .............................................. 242
8.20.7 TFPObjectList.Add ................................................ 242
8.20.8 TFPObjectList.Delete .............................................. 243
8.20.9 TFPObjectList.Exchange ........................................... 243
8.20.10 TFPObjectList.Expand ............................................ 243
8.20.11 TFPObjectList.Extract ............................................ 244
8.20.12 TFPObjectList.Remove ............................................ 244
8.20.13 TFPObjectList.IndexOf ........................................ 244
8.20.14 TFPObjectList.FindInstanceOf ................................. 244
8.20.15 TFPObjectList.Insert ........................................... 245
8.20.16 TFPObjectList.First ............................................ 245
8.20.17 TFPObjectList.Last ............................................ 245
8.20.18 TFPObjectList.Move ............................................ 246
8.20.19 TFPObjectList.Assign .......................................... 246
8.20.20 TFPObjectList.Pack ............................................ 246
8.20.21 TFPObjectList.Sort ............................................ 246
8.20.22 TFPObjectList.ForEachCall ................................. 247
8.20.23 TFPObjectList.Capacity ................................... 247
8.20.24 TFPObjectList.Count ........................................ 247
8.20.25 TFPObjectList.OwnsObjects ................................. 248
8.20.26 TFPObjectList.Items ......................................... 248
8.20.27 TFPObjectList.List ........................................... 248
8.21 TFPStringHashTable ................................................ 248
8.21.1 Description ................................................... 248
8.21.2 Method overview ............................................... 249
8.21.3 Property overview ............................................. 249
8.21.4 TFPStringHashTable.Iterate ............................... 249
8.21.5 TFPStringHashTable.Add .................................... 249
8.21.6 TFPStringHashTable.Items .................................. 249
8.22 THTCustomNode ....................................................... 250
8.22.1 Description ................................................... 250
8.22.2 Method overview ............................................... 250
8.22.3 Property overview ............................................. 250
8.22.4 THTCustomNode.CreateWith ............................... 250
8.22.5 THTCustomNode.HasKey .................................... 250
8.22.6 THTCustomNode.Key ........................................ 251
8.23 THTDataNode .......................................................... 251
8.23.1 Description ................................................... 251
8.23.2 Property overview ............................................. 251
8.23.3 THTDataNode.Data ........................................... 251
8.24 THTObjectNode ........................................................ 251
8.24.1 Description ................................................... 251
8.24.2 Property overview ............................................. 252
8.24.3 THTObjectNode.Data ........................................ 252
8.25 THTOwnedObjectNode ............................................... 252
8.25.1 Description ................................................... 252
8.25.2 Method overview ............................................. 252
8.25.3 THTOwnedObjectNode.Destroy .......................... 252
8.26 THTStringNode ................................................. 252
  8.26.1 Description ............................................... 252
  8.26.2 Property overview ........................................ 253
  8.26.3 THTStringNode.Data ..................................... 253
8.27 TObjectBucketList ............................................. 253
  8.27.1 Description ............................................... 253
  8.27.2 Method overview ........................................ 253
  8.27.3 Property overview ........................................ 253
  8.27.4 TObjectBucketList.Add .................................. 253
  8.27.5 TObjectBucketList.Remove ............................. 254
  8.27.6 TObjectBucketList.Data ................................ 254
8.28 TObjectList .................................................... 254
  8.28.1 Description ............................................... 254
  8.28.2 Method overview ........................................ 254
  8.28.3 Property overview ........................................ 255
  8.28.4 TObjectList.Create ..................................... 255
  8.28.5 TObjectList.Add ........................................ 255
  8.28.6 TObjectList.Extract .................................... 255
  8.28.7 TObjectList.Remove .................................... 256
  8.28.8 TObjectList.IndexOf ................................... 256
  8.28.9 TObjectList.FindInstanceOf ......................... 256
  8.28.10 TObjectList.Insert .................................... 257
  8.28.11 TObjectList.First .................................... 257
  8.28.12 TObjectList.Last ..................................... 257
  8.28.13 TObjectList.OwnsObjects ......................... 257
  8.28.14 TObjectList.Items ................................... 258
8.29 TObjectQueue .................................................. 258
  8.29.1 Method overview ........................................ 258
  8.29.2 TObjectQueue.Push .................................... 258
  8.29.3 TObjectQueue.Pop ..................................... 258
  8.29.4 TObjectQueue.Peek .................................... 259
8.30 TObjectStack .................................................. 259
  8.30.1 Description ............................................... 259
  8.30.2 Method overview ........................................ 259
  8.30.3 TObjectStack.Push .................................... 259
  8.30.4 TObjectStack.Pop ..................................... 259
  8.30.5 TObjectStack.Peek ................................... 260
8.31 TOrderedList ................................................... 260
CONTENTS

9.4.20 TCustomApplication.HelpFile ................................................. 272
9.4.21 TCustomApplication.Terminated ................................................. 272
9.4.22 TCustomApplication.Title ........................................................ 272
9.4.23 TCustomApplication.OnException ............................................... 272
9.4.24 TCustomApplication.ConsoleApplication ...................................... 273
9.4.25 TCustomApplication.Location .................................................. 273
9.4.26 TCustomApplication.Params ...................................................... 273
9.4.27 TCustomApplication.ParamCount ............................................... 274
9.4.28 TCustomApplication.EnvironmentVariable ................................... 274
9.4.29 TCustomApplication.OptionChar ............................................... 274
9.4.30 TCustomApplication.CaseSensitiveOptions .................................. 274
9.4.31 TCustomApplication.StopOnException ....................................... 275
9.4.32 TCustomApplication.ExceptionExitCode ..................................... 275
9.4.33 TCustomApplication.EventLogFilter ......................................... 275
9.4.34 TCustomApplication.SingleInstance ......................................... 276
9.4.35 TCustomApplication.SingleInstanceClass ................................... 276
9.4.36 TCustomApplication.SingleInstanceEnabled .................................. 276

10 Reference for unit 'daemonapp' ......................................................... 277
10.1 Used units ................................................................................. 277
10.2 Overview ................................................................................... 277
10.3 Constants, types and variables ..................................................... 278
  10.3.1 Resource strings ................................................................. 278
  10.3.2 Types ................................................................................ 279
  10.3.3 Variables ........................................................................... 283
10.4 Procedures and functions ............................................................. 283
  10.4.1 Application ....................................................................... 283
  10.4.2 DaemonError ..................................................................... 283
  10.4.3 RegisterDaemonApplicationClass ........................................ 284
  10.4.4 RegisterDaemonClass .......................................................... 284
  10.4.5 RegisterDaemonMapper ......................................................... 284
10.5 EDaemon ....................................................................................... 284
  10.5.1 Description ....................................................................... 284
10.6 TCustomDaemon ......................................................................... 285
  10.6.1 Description ....................................................................... 285
  10.6.2 Method overview ............................................................... 285
  10.6.3 Property overview ............................................................... 285
  10.6.4 TCustomDaemon.CheckControlMessages .................................. 285
  10.6.5 TCustomDaemon.LogMessage ............................................... 285
  10.6.6 TCustomDaemon.ReportStatus ............................................. 286

21
| 10.6.7 | TCustomDaemon.Definition | 286 |
| 10.6.8 | TCustomDaemon.DaemonThread | 286 |
| 10.6.9 | TCustomDaemon.Controller | 287 |
| 10.6.10 | TCustomDaemon.Status | 287 |
| 10.6.11 | TCustomDaemon.Logger | 287 |
| 10.7 | TCustomDaemonApplication | 287 |
| 10.7.1 | Description | 287 |
| 10.7.2 | Method overview | 288 |
| 10.7.3 | Property overview | 288 |
| 10.7.4 | TCustomDaemonApplication.Create | 288 |
| 10.7.5 | TCustomDaemonApplication.Destroy | 288 |
| 10.7.6 | TCustomDaemonApplication.ShowException | 288 |
| 10.7.7 | TCustomDaemonApplication.CreateDaemon | 289 |
| 10.7.8 | TCustomDaemonApplication.StopDaemons | 289 |
| 10.7.9 | TCustomDaemonApplication.InstallDaemons | 289 |
| 10.7.10 | TCustomDaemonApplication.RunDaemons | 289 |
| 10.7.11 | TCustomDaemonApplication.UnInstallDaemons | 290 |
| 10.7.12 | TCustomDaemonApplication.ShowHelp | 290 |
| 10.7.13 | TCustomDaemonApplication.CreateForm | 290 |
| 10.7.14 | TCustomDaemonApplication.OnRun | 290 |
| 10.7.15 | TCustomDaemonApplication.EventLog | 291 |
| 10.7.16 | TCustomDaemonApplication.GUIMainLoop | 291 |
| 10.7.17 | TCustomDaemonApplication.GuiHandle | 291 |
| 10.7.18 | TCustomDaemonApplication.RunMode | 291 |
| 10.7.19 | TCustomDaemonApplication.AutoRegisterMessageFile | 292 |
| 10.8 | TCustomDaemonMapper | 292 |
| 10.8.1 | Description | 292 |
| 10.8.2 | Method overview | 292 |
| 10.8.3 | Property overview | 292 |
| 10.8.4 | TCustomDaemonMapper.Create | 292 |
| 10.8.5 | TCustomDaemonMapper.Destroy | 293 |
| 10.8.6 | TCustomDaemonMapper.DaemonDefs | 293 |
| 10.8.7 | TCustomDaemonMapper.onCreate | 293 |
| 10.8.8 | TCustomDaemonMapper.onDestroy | 294 |
| 10.8.9 | TCustomDaemonMapper.OnRun | 294 |
| 10.8.10 | TCustomDaemonMapper.OnInstall | 294 |
| 10.8.11 | TCustomDaemonMapper.OnUnInstall | 294 |
| 10.9 | TDaemon | 295 |
| 10.9.1 | Description | 295 |
| 10.9.2 | Property overview | 295 |
10.16.5 TDependencies.Items ................................................. 312
10.17 TDependency ................................................................. 313
  10.17.1 Description ............................................................ 313
  10.17.2 Method overview .................................................... 313
  10.17.3 Property overview .................................................. 313
  10.17.4 TDependency.Assign .............................................. 313
  10.17.5 TDependency.Name .................................................. 313
  10.17.6 TDependency.IsGroup ............................................. 313
10.18 TWinBindings ................................................................. 314
  10.18.1 Description ............................................................ 314
  10.18.2 Method overview .................................................... 314
  10.18.3 Property overview .................................................. 314
  10.18.4 TWinBindings.Create .............................................. 314
  10.18.5 TWinBindings.Destroy ............................................. 314
  10.18.6 TWinBindings.Assign ............................................. 315
  10.18.7 TWinBindings.ErrCode ............................................ 315
  10.18.8 TWinBindings.Win32ErrCode .................................... 315
  10.18.9 TWinBindings.Dependencies ..................................... 315
  10.18.10 TWinBindings.GroupName ....................................... 316
  10.18.11 TWinBindings.Password ......................................... 316
  10.18.12 TWinBindings.UserName ......................................... 316
  10.18.13 TWinBindings.StartType ....................................... 316
  10.18.14 TWinBindings.WaitHint ........................................ 317
  10.18.15 TWinBindings.IDTag ............................................ 317
  10.18.16 TWinBindings.ServiceType ..................................... 317
  10.18.17 TWinBindings.ErrorSeverity ................................... 317
  10.18.18 TWinBindings.AcceptedCodes .................................. 318

11 Reference for unit 'DB' .................................................. 319
  11.1 Used units ...................................................................... 319
  11.2 Overview ...................................................................... 319
  11.3 Constants, types and variables .................................... 319
    11.3.1 Constants .................................................................. 319
    11.3.2 Types .................................................................... 321
    11.3.3 Variables .................................................................. 335
  11.4 Procedures and functions ............................................. 336
    11.4.1 BuffersEqual .......................................................... 336
    11.4.2 DatabaseError ........................................................ 336
    11.4.3 DatabaseErrorFmt .................................................... 336
    11.4.4 DateTimeRecToDateTime .......................................... 337
11.4.5 DateTimeToDateTimeRec ........................................... 337
11.4.6 DisposeMem ..................................................... 337
11.4.7 enumerator(TDataSet):TDataSetEnumerator ...................... 337
11.4.8 ExtractFieldName ................................................. 338
11.4.9 SkipComments ..................................................... 338
11.5 TLookupListRec ..................................................... 338
11.6 EDatabaseError ..................................................... 338
11.6.1 Description ....................................................... 338
11.7 EUpdateError ....................................................... 339
11.7.1 Description ....................................................... 339
11.7.2 Method overview ................................................ 339
11.7.3 Property overview .............................................. 339
11.7.4 EUpdateError.Create .......................................... 339
11.7.5 EUpdateError.Destroy .......................................... 339
11.7.6 EUpdateError.Context .......................................... 340
11.7.7 EUpdateError.ErrorCode ...................................... 340
11.7.8 EUpdateError.OriginalException ............................... 340
11.7.9 EUpdateError.PreviousError .................................... 340
11.8 IProviderSupport ................................................... 341
11.8.1 Description ....................................................... 341
11.8.2 Method overview ................................................ 341
11.8.3 IProviderSupport.PSEndTransaction ............................ 341
11.8.4 IProviderSupport.PSExecute ................................... 341
11.8.5 IProviderSupport.PSExecuteStatement .......................... 342
11.8.6 IProviderSupport.PSGetAttributes ............................. 342
11.8.7 IProviderSupport.PSGetCommandText ........................... 342
11.8.8 IProviderSupport.PSGetCommandType ........................... 343
11.8.9 IProviderSupport.PSGetDefaultOrder ......................... 343
11.8.10 IProviderSupport.PSGetIndexDefs ............................ 343
11.8.11 IProviderSupport.PSGetKeyFields ............................ 343
11.8.12 IProviderSupport.PSGetParams ............................... 344
11.8.13 IProviderSupport.PSGetQuoteChar ............................ 344
11.8.14 IProviderSupport.PSGetTableName ............................ 344
11.8.15 IProviderSupport.PSGetUpdateException ....................... 344
11.8.16 IProviderSupport.PSInTransaction ........................... 345
11.8.17 IProviderSupport.PSIsSQLBased ............................... 345
11.8.18 IProviderSupport.PSIsSQLSupported .......................... 345
11.8.19 IProviderSupport.PSReset ..................................... 345
11.8.20 IProviderSupport.PSSetCommandText ........................ 346
11.8.21 IProviderSupport.PSSetParams ............................... 346
11.8.22 IProviderSupport.PSStartTransaction .......................... 346
11.8.23 IProviderSupport.PSUpdateRecord ............................ 346
11.9 TAutoIncField .................................................. 347
  11.9.1 Description ................................................. 347
  11.9.2 Method overview .......................................... 347
  11.9.3 TAutoIncField.Create ..................................... 347
11.10 TBCDField ..................................................... 347
  11.10.1 Description ................................................. 347
  11.10.2 Method overview .......................................... 347
  11.10.3 Property overview ....................................... 348
  11.10.4 TBCDField.Create ......................................... 348
  11.10.5 TBCDField.CheckRange .................................... 348
  11.10.6 TBCDField.Value .......................................... 348
  11.10.7 TBCDField.Precision ...................................... 349
  11.10.8 TBCDField.Currency ....................................... 349
  11.10.9 TBCDField.MaxValue ..................................... 349
  11.10.10 TBCDField.MinValue ..................................... 350
  11.10.11 TBCDField.Size ......................................... 350
11.11 TBinaryField .................................................. 350
  11.11.1 Description ................................................. 350
  11.11.2 Method overview .......................................... 350
  11.11.3 Property overview ....................................... 350
  11.11.4 TBinaryField.Create ..................................... 351
  11.11.5 TBinaryField.Size ....................................... 351
11.12 TBlobField ..................................................... 351
  11.12.1 Description ................................................. 351
  11.12.2 Method overview .......................................... 351
  11.12.3 Property overview ....................................... 352
  11.12.4 TBlobField.Create ......................................... 352
  11.12.5 TBlobField.Clear ......................................... 352
  11.12.6 TBlobField.IsBlob ........................................ 352
  11.12.7 TBlobField.LoadFromFile .................................. 352
  11.12.8 TBlobField.LoadFromStream ................................ 353
  11.12.9 TBlobField.SaveToFile .................................... 353
  11.12.10 TBlobField.SaveToStream ................................ 353
  11.12.11 TBlobField.SetFieldType ................................ 354
  11.12.12 TBlobField.BlobSize .................................... 354
  11.12.13 TBlobField.Modified ................................... 354
  11.12.14 TBlobField.Value ....................................... 354
  11.12.15 TBlobField.Transliterate ................................ 355
CONTENTS

11.18.5 TCustomConnection.Destroy ........................................ 362
11.18.6 TCustomConnection.Open ........................................... 362
11.18.7 TCustomConnection.DataSetCount ................................. 363
11.18.8 TCustomConnection.DataSets ...................................... 363
11.18.9 TCustomConnection.Connected ................................... 363
11.18.10 TCustomConnection.LoginPrompt ................................. 364
11.18.11 TCustomConnection.AfterConnect ................................ 364
11.18.12 TCustomConnection.AfterDisconnect ............................ 364
11.18.13 TCustomConnection.BeforeConnect ............................... 365
11.18.14 TCustomConnection.BeforeDisconnect ............................ 365
11.18.15 TCustomConnection.OnLogin ...................................... 365

11.19 TDatabase .............................................................. 365
11.19.1 Description ......................................................... 365
11.19.2 Method overview .................................................. 366
11.19.3 Property overview .................................................. 366
11.19.4 TDatabase.Create .................................................. 366
11.19.5 TDatabase.Destroy .................................................. 366
11.19.6 TDatabase.CloseDataSets .......................................... 367
11.19.7 TDatabase.CloseTransactions .................................... 367
11.19.8 TDatabase.StartTransaction ....................................... 367
11.19.9 TDatabase.EndTransaction ....................................... 367
11.19.10 TDatabase.TransactionCount .................................... 368
11.19.11 TDatabase.Transactions ......................................... 368
11.19.12 TDatabase.Directory ............................................. 368
11.19.13 TDatabase.IsSQLBased ............................................ 368
11.19.14 TDatabase.ConnectionString ................................. 369
11.19.15 TDatabase.DatabaseName ....................................... 369
11.19.16 TDatabase.KeepConnection ..................................... 369
11.19.17 TDatabase.Params ................................................ 369

11.20 TDataLink .............................................................. 370
11.20.1 Description ......................................................... 370
11.20.2 Method overview .................................................. 370
11.20.3 Property overview .................................................. 370
11.20.4 TDataLink.Create .................................................. 370
11.20.5 TDataLink.Destroy .................................................. 371
11.20.6 TDataLink.Edit ..................................................... 371
11.20.7 TDataLink.UpdateRecord .......................................... 371
11.20.8 TDataLink.ExecuteAction ......................................... 371
11.20.9 TDataLink.UpdateAction .......................................... 372
11.20.10 TDataLink.Active .................................................. 372
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>11.20.1 TDataLink.ActiveRecord</td>
<td>372</td>
</tr>
<tr>
<td>11.20.12 TDataLink.BOF</td>
<td>372</td>
</tr>
<tr>
<td>11.20.13 TDataLink.BufferCount</td>
<td>373</td>
</tr>
<tr>
<td>11.20.14 TDataLink.DataSet</td>
<td>373</td>
</tr>
<tr>
<td>11.20.15 TDataLink.DataSource</td>
<td>373</td>
</tr>
<tr>
<td>11.20.16 TDataLink.DataSourceFixed</td>
<td>373</td>
</tr>
<tr>
<td>11.20.17 TDataLink.Editing</td>
<td>374</td>
</tr>
<tr>
<td>11.20.18 TDataLink.Eof</td>
<td>374</td>
</tr>
<tr>
<td>11.20.19 TDataLink.ReadOnly</td>
<td>374</td>
</tr>
<tr>
<td>11.20.20 TDataLink.RecordCount</td>
<td>374</td>
</tr>
<tr>
<td>11.21 TDataSet</td>
<td>375</td>
</tr>
<tr>
<td>11.21.1 Description</td>
<td>375</td>
</tr>
<tr>
<td>11.21.2 Method overview</td>
<td>377</td>
</tr>
<tr>
<td>11.21.3 Property overview</td>
<td>378</td>
</tr>
<tr>
<td>11.21.4 TDataSet.Create</td>
<td>379</td>
</tr>
<tr>
<td>11.21.5 TDataSet.Destroy</td>
<td>379</td>
</tr>
<tr>
<td>11.21.6 TDataSet.ActiveBuffer</td>
<td>379</td>
</tr>
<tr>
<td>11.21.7 TDataSet.GetFieldData</td>
<td>379</td>
</tr>
<tr>
<td>11.21.8 TDataSet.SetFieldData</td>
<td>380</td>
</tr>
<tr>
<td>11.21.9 TDataSet.Append</td>
<td>380</td>
</tr>
<tr>
<td>11.21.10 TDataSet.AppendRecord</td>
<td>380</td>
</tr>
<tr>
<td>11.21.11 TDataSet.BookmarkValid</td>
<td>381</td>
</tr>
<tr>
<td>11.21.12 TDataSet.Cancel</td>
<td>381</td>
</tr>
<tr>
<td>11.21.13 TDataSet.CheckBrowseMode</td>
<td>381</td>
</tr>
<tr>
<td>11.21.14 TDataSet.ClearFields</td>
<td>381</td>
</tr>
<tr>
<td>11.21.15 TDataSet.Close</td>
<td>382</td>
</tr>
<tr>
<td>11.21.16 TDataSet.ControlsDisabled</td>
<td>382</td>
</tr>
<tr>
<td>11.21.17 TDataSet.CompareBookmarks</td>
<td>382</td>
</tr>
<tr>
<td>11.21.18 TDataSet.CreateBlobStream</td>
<td>383</td>
</tr>
<tr>
<td>11.21.19 TDataSet.CursorPosChanged</td>
<td>383</td>
</tr>
<tr>
<td>11.21.20 TDataSet.DataConvert</td>
<td>383</td>
</tr>
<tr>
<td>11.21.21 TDataSet.Delete</td>
<td>383</td>
</tr>
<tr>
<td>11.21.22 TDataSet.DisableControls</td>
<td>384</td>
</tr>
<tr>
<td>11.21.23 TDataSet.Edit</td>
<td>384</td>
</tr>
<tr>
<td>11.21.24 TDataSet.EnableControls</td>
<td>385</td>
</tr>
<tr>
<td>11.21.25 TDataSet.FieldName</td>
<td>385</td>
</tr>
<tr>
<td>11.21.26 TDataSet.FindField</td>
<td>385</td>
</tr>
<tr>
<td>11.21.27 TDataSet.FindFirst</td>
<td>386</td>
</tr>
<tr>
<td>11.21.28 TDataSet.FindLast</td>
<td>386</td>
</tr>
<tr>
<td>11.21.29 TDataSet.FindNext</td>
<td>386</td>
</tr>
</tbody>
</table>
11.21.70 TDataSet.RecordCount ........................................... 400
11.21.71 TDataSet.RecNo .................................................. 400
11.21.72 TDataSet.RecordSize ............................................ 400
11.21.73 TDataSet.State .................................................. 401
11.21.74 TDataSet.Fields ............................................... 401
11.21.75 TDataSet.FieldValues ........................................ 401
11.21.76 TDataSet.Filter ................................................ 402
11.21.77 TDataSet.Filtered .............................................. 402
11.21.78 TDataSet.FilterOptions ...................................... 402
11.21.79 TDataSet.Active ............................................... 403
11.21.80 TDataSet.AutoCalcFields .................................... 403
11.21.81 TDataSet.BeforeOpen .......................................... 403
11.21.82 TDataSet.AfterOpen .......................................... 404
11.21.83 TDataSet.BeforeClose ....................................... 404
11.21.84 TDataSet.AfterClose ......................................... 404
11.21.85 TDataSet.BeforeInsert ....................................... 404
11.21.86 TDataSet.AfterInsert ....................................... 405
11.21.87 TDataSet.BeforeEdit ......................................... 405
11.21.88 TDataSet.AfterEdit .......................................... 405
11.21.89 TDataSet.BeforePost ......................................... 406
11.21.90 TDataSet.AfterPost .......................................... 406
11.21.91 TDataSet.BeforeCancel ..................................... 406
11.21.92 TDataSet.AfterCancel ....................................... 407
11.21.93 TDataSet.BeforeDelete ...................................... 407
11.21.94 TDataSet.AfterDelete ....................................... 407
11.21.95 TDataSet.BeforeScroll ..................................... 407
11.21.96 TDataSet.AfterScroll ...................................... 408
11.21.97 TDataSet.BeforeRefresh .................................... 408
11.21.98 TDataSet.AfterRefresh ...................................... 408
11.21.99 TDataSet.OnCalcFields ..................................... 409
11.21.100 TDataSet.OnDeleteError .................................. 409
11.21.101 TDataSet.OnEditError ..................................... 409
11.21.102 TDataSet.OnFilterRecord ................................ 410
11.21.103 TDataSet.OnNewRecord .................................. 410
11.21.104 TDataSet.OnPostError .................................... 411

11.22 TDataSetEnumerator ............................................. 411
11.22.1 Description ..................................................... 411
11.22.2 Method overview ............................................. 411
11.22.3 Property overview ........................................... 412
11.22.4 TDataSetEnumerator.Create ................................. 412
11.27.4 TDBTransaction.Create ........................................... 420
11.27.5 TDBTransaction.Destroy ........................................ 421
11.27.6 TDBTransaction.CloseDataSets ................................. 421
11.27.7 TDBTransaction.DataBase ...................................... 421
11.27.8 TDBTransaction.Active ....................................... 421
11.28 TDefCollection ..................................................... 422
  11.28.1 Description ................................................... 422
  11.28.2 Method overview ............................................. 422
  11.28.3 Property overview .......................................... 422
  11.28.4 TDefCollection.create ..................................... 422
  11.28.5 TDefCollection.Find ....................................... 422
  11.28.6 TDefCollection.GetItemNames .............................. 423
  11.28.7 TDefCollection.IndexOf .................................. 423
  11.28.8 TDefCollection.Dataset ................................... 423
  11.28.9 TDefCollection.Updated .................................. 423
11.29 TDetailDataLink .................................................. 424
  11.29.1 Description ................................................... 424
  11.29.2 Property overview .......................................... 424
  11.29.3 TDetailDataLink.DetailDataSet ............................ 424
11.30 TField ............................................................ 424
  11.30.1 Description ................................................... 424
  11.30.2 Method overview ............................................ 425
  11.30.3 Property overview .......................................... 427
  11.30.4 TField.Create ............................................... 428
  11.30.5 TField.Destroy ............................................... 428
  11.30.6 TField.Assign ............................................... 428
  11.30.7 TField.AssignValue ....................................... 428
  11.30.8 TField.Clear ................................................. 429
  11.30.9 TField.FocusControl ....................................... 429
  11.30.10 TField.GetData ............................................. 429
  11.30.11 TField.IsBlob .............................................. 430
  11.30.12 TField.IsValidChar ....................................... 430
  11.30.13 TField.RefreshLookupList ................................. 430
  11.30.14 TField.SetData ............................................. 430
  11.30.15 TField.SetFieldType ..................................... 431
  11.30.16 TField.Validate ........................................... 431
  11.30.17 TField.AsBCD .............................................. 431
  11.30.18 TField.AsBoolean ....................................... 432
  11.30.19 TField.AsBytes ............................................ 432
  11.30.20 TField.AsCurrency ....................................... 432
<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>11.44.6</td>
<td>TLookupList.Clear</td>
<td>480</td>
</tr>
<tr>
<td>11.44.7</td>
<td>TLookupList.FirstKeyByValue</td>
<td>480</td>
</tr>
<tr>
<td>11.44.8</td>
<td>TLookupList.ValueOfKey</td>
<td>481</td>
</tr>
<tr>
<td>11.44.9</td>
<td>TLookupList.ValuesToStrings</td>
<td>481</td>
</tr>
<tr>
<td>11.45</td>
<td>TMasterDataLink</td>
<td>481</td>
</tr>
<tr>
<td>11.45.1</td>
<td>Description</td>
<td>481</td>
</tr>
<tr>
<td>11.45.2</td>
<td>Method overview</td>
<td>481</td>
</tr>
<tr>
<td>11.45.3</td>
<td>Property overview</td>
<td>482</td>
</tr>
<tr>
<td>11.45.4</td>
<td>TMasterDataLink.Create</td>
<td>482</td>
</tr>
<tr>
<td>11.45.5</td>
<td>TMasterDataLink.Destroy</td>
<td>482</td>
</tr>
<tr>
<td>11.45.6</td>
<td>TMasterDataLink.FieldNames</td>
<td>482</td>
</tr>
<tr>
<td>11.45.7</td>
<td>TMasterDataLink.Fields</td>
<td>483</td>
</tr>
<tr>
<td>11.45.8</td>
<td>TMasterDataLink.OnMasterChange</td>
<td>483</td>
</tr>
<tr>
<td>11.45.9</td>
<td>TMasterDataLink.OnMasterDisable</td>
<td>483</td>
</tr>
<tr>
<td>11.46</td>
<td>TMasterParamsDataLink</td>
<td>483</td>
</tr>
<tr>
<td>11.46.1</td>
<td>Description</td>
<td>483</td>
</tr>
<tr>
<td>11.46.2</td>
<td>Method overview</td>
<td>484</td>
</tr>
<tr>
<td>11.46.3</td>
<td>Property overview</td>
<td>484</td>
</tr>
<tr>
<td>11.46.4</td>
<td>TMasterParamsDataLink.Create</td>
<td>484</td>
</tr>
<tr>
<td>11.46.5</td>
<td>TMasterParamsDataLink.RefreshParamNames</td>
<td>484</td>
</tr>
<tr>
<td>11.46.6</td>
<td>TMasterParamsDataLink.CopyParamsFromMaster</td>
<td>484</td>
</tr>
<tr>
<td>11.46.7</td>
<td>TMasterParamsDataLink.Params</td>
<td>485</td>
</tr>
<tr>
<td>11.47</td>
<td>TMemoField</td>
<td>485</td>
</tr>
<tr>
<td>11.47.1</td>
<td>Description</td>
<td>485</td>
</tr>
<tr>
<td>11.47.2</td>
<td>Method overview</td>
<td>485</td>
</tr>
<tr>
<td>11.47.3</td>
<td>Property overview</td>
<td>485</td>
</tr>
<tr>
<td>11.47.4</td>
<td>TMemoField.Create</td>
<td>485</td>
</tr>
<tr>
<td>11.47.5</td>
<td>TMemoField.CodePage</td>
<td>486</td>
</tr>
<tr>
<td>11.47.6</td>
<td>TMemoField.Transliterate</td>
<td>486</td>
</tr>
<tr>
<td>11.48</td>
<td>TNamedItem</td>
<td>486</td>
</tr>
<tr>
<td>11.48.1</td>
<td>Description</td>
<td>486</td>
</tr>
<tr>
<td>11.48.2</td>
<td>Property overview</td>
<td>486</td>
</tr>
<tr>
<td>11.48.3</td>
<td>TNamedItem.DisplayName</td>
<td>486</td>
</tr>
<tr>
<td>11.48.4</td>
<td>TNamedItem.Name</td>
<td>487</td>
</tr>
<tr>
<td>11.49</td>
<td>TNumericField</td>
<td>487</td>
</tr>
<tr>
<td>11.49.1</td>
<td>Description</td>
<td>487</td>
</tr>
<tr>
<td>11.49.2</td>
<td>Method overview</td>
<td>487</td>
</tr>
<tr>
<td>11.49.3</td>
<td>Property overview</td>
<td>487</td>
</tr>
<tr>
<td>11.49.4</td>
<td>TNumericField.Create</td>
<td>487</td>
</tr>
<tr>
<td>11.49.5</td>
<td>TNumericField.Alignment</td>
<td>488</td>
</tr>
</tbody>
</table>
11.60.2 Method overview .......................... 515
11.60.3 TWordField.Create ....................... 515

12 Reference for unit 'dbugintf' 516
12.1 Used units .................................. 516
12.2 Overview ................................... 516
12.3 Writing a debug server ...................... 516
12.4 Constants, types and variables .......... 517
   12.4.1 Resource strings ...................... 517
   12.4.2 Types ................................ 517
   12.4.3 Variables ............................. 517
12.5 Procedures and functions ............... 518
   12.5.1 FreeDebugClient ...................... 518
   12.5.2 GetDebuggingEnabled .................. 518
   12.5.3 InitDebugClient ....................... 518
   12.5.4 SendBoolean .......................... 519
   12.5.5 SendDateTime ......................... 519
   12.5.6 SendDebug ............................ 519
   12.5.7 SendDebugEx .......................... 519
   12.5.8 SendDebugFmt ......................... 520
   12.5.9 SendDebugFmtEx ....................... 520
   12.5.10 SendInteger ......................... 520
   12.5.11 SendMethodEnter ..................... 521
   12.5.12 SendMethodExit ...................... 521
   12.5.13 SendPointer ........................ 521
   12.5.14 SendSeparator ....................... 522
   12.5.15 SetDebuggingEnabled ................. 522
   12.5.16 StartDebugServer .................... 522

13 Reference for unit 'dbugmsg' 523
13.1 Used units ................................ 523
13.2 Overview .................................. 523
13.3 Constants, types and variables .......... 523
   13.3.1 Constants ....................... 523
   13.3.2 Types .............................. 524
13.4 Procedures and functions ............... 524
   13.4.1 DebugMessageName .................... 524
   13.4.2 ReadDebugMessageFromStream .......... 524
   13.4.3 WriteDebugMessageToStream .......... 524
13.5 TDebugMessage .......................... 525
## 14 Reference for unit 'eventlog'

14.1 Used units .............................................. 526
14.2 Overview ................................................ 526
14.3 Constants, types and variables ......................... 526
   14.3.1 Resource strings ................................ 526
   14.3.2 Types .............................................. 527
14.4 ELogError ............................................... 528
   14.4.1 Description ....................................... 528
14.5 TEventLog ................................................ 528
   14.5.1 Description ....................................... 528
   14.5.2 Method overview ................................... 528
   14.5.3 Property overview ................................ 529
   14.5.4 TEventLog.Destroy ................................ 529
   14.5.5 TEventLog.EventTypeToString ..................... 529
   14.5.6 TEventLog.RegisterMessageFile ................... 529
   14.5.7 TEventLog.UnRegisterMessageFile ................. 530
   14.5.8 TEventLog.Pause ................................... 531
   14.5.9 TEventLog.Resume .................................. 531
   14.5.10 TEventLog.Log .................................... 531
   14.5.11 TEventLog.Warning ................................ 531
   14.5.12 TEventLog.Error .................................. 532
   14.5.13 TEventLog.Debug .................................. 532
   14.5.14 TEventLog.Info ................................... 532
   14.5.15 TEventLog.AppendContent ......................... 532
   14.5.16 TEventLog.Identification ......................... 533
   14.5.17 TEventLog.LogType ................................ 533
   14.5.18 TEventLog.Active .................................. 533
   14.5.19 TEventLog.RaiseExceptionOnError ................. 534
   14.5.20 TEventLog.DefaultEventType ....................... 534
   14.5.21 TEventLog.FileName ................................ 534
   14.5.22 TEventLog.TimeStampFormat ....................... 534
   14.5.23 TEventLog.CustomLogType ......................... 535
   14.5.24 TEventLog.EventIDOffset ......................... 535
   14.5.25 TEventLog.OnGetCustomCategory ................... 535
   14.5.26 TEventLog.OnGetCustomEventID .................... 536
   14.5.27 TEventLog.OnGetCustomEvent ...................... 536
   14.5.28 TEventLog.OnLogMessage ......................... 536
   14.5.29 TEventLog.Paused .................................. 536

15 Reference for unit 'ezcgi'  

45
16.4.9 JSONTypeName ................................................. 552
16.4.10 SetJSONInstanceType ........................................ 552
16.4.11 SetJSONParserHandler ........................................ 553
16.4.12 SetJSONStringParserHandler ................................. 553
16.4.13 StringToJSONString ......................................... 553
16.5 TJSONEnum ..................................................... 554
16.6 EJSON .......................................................... 554
16.6.1 Description .................................................. 554
16.7 TBaseJSONEnumerator ............................................. 554
16.7.1 Description .................................................. 554
16.7.2 Method overview ............................................. 554
16.7.3 Property overview ........................................... 554
16.7.4 TBaseJSONEnumerator.GetCurrent ......................... 554
16.7.5 TBaseJSONEnumerator.MoveNext ......................... 555
16.7.6 TBaseJSONEnumerator.Current ............................ 555
16.8 TJSONArray ................................................... 555
16.8.1 Description .................................................. 555
16.8.2 Method overview ............................................. 556
16.8.3 Property overview ........................................... 556
16.8.4 TJSONArray.Create ........................................ 556
16.8.5 TJSONArray.Destroy ........................................ 557
16.8.6 TJSONArray.JSONType ...................................... 557
16.8.7 TJSONArray.Clone .......................................... 557
16.8.8 TJSONArray.Iterate ........................................ 557
16.8.9 TJSONArray.IndexOf ...................................... 558
16.8.10 TJSONArray.GetEnumerator ................................. 558
16.8.11 TJSONArray.Clear ......................................... 558
16.8.12 TJSONArray.Add .......................................... 558
16.8.13 TJSONArray.Delete ........................................ 559
16.8.14 TJSONArray.Exchange .................................... 559
16.8.15 TJSONArray.Extract ...................................... 559
16.8.16 TJSONArray.Insert ........................................ 560
16.8.17 TJSONArray.Move ......................................... 560
16.8.18 TJSONArray.Remove ....................................... 560
16.8.19 TJSONArray.Sort .......................................... 561
16.8.20 TJSONArray.Items ......................................... 561
16.8.21 TJSONArray.Types ........................................ 561
16.8.22 TJSONArray.Nulls ........................................ 561
16.8.23 TJSONArray.Integers ..................................... 562
16.8.24 TJSONArray.Int64s ....................................... 562
<p>| 16.10.25 | TJSONData.IsNull | 576 |
| 16.10.26 | TJSONData.AsJSON | 576 |
| 16.11 | TJSONFloatNumber | 577 |
| 16.11.1 | Description | 577 |
| 16.11.2 | Method overview | 577 |
| 16.11.3 | TJSONFloatNumber.Create | 577 |
| 16.11.4 | TJSONFloatNumber.NumberType | 577 |
| 16.11.5 | TJSONFloatNumber.Clear | 578 |
| 16.11.6 | TJSONFloatNumber.Clone | 578 |
| 16.12 | TJSONInt64Number | 578 |
| 16.12.1 | Description | 578 |
| 16.12.2 | Method overview | 578 |
| 16.12.3 | TJSONInt64Number.Create | 578 |
| 16.12.4 | TJSONInt64Number.NumberType | 579 |
| 16.12.5 | TJSONInt64Number.Clear | 579 |
| 16.12.6 | TJSONInt64Number.Clone | 579 |
| 16.13 | TJSONIntegerNumber | 579 |
| 16.13.1 | Description | 579 |
| 16.13.2 | Method overview | 579 |
| 16.13.3 | TJSONIntegerNumber.Create | 580 |
| 16.13.4 | TJSONIntegerNumber.NumberType | 580 |
| 16.13.5 | TJSONIntegerNumber.Clear | 580 |
| 16.13.6 | TJSONIntegerNumber.Clone | 580 |
| 16.14 | TJSONNull | 580 |
| 16.14.1 | Description | 580 |
| 16.14.2 | Method overview | 581 |
| 16.14.3 | TJSONNull.JSONType | 581 |
| 16.14.5 | TJSONNull.Clone | 581 |
| 16.15 | TJSONNumber | 581 |
| 16.15.1 | Description | 581 |
| 16.15.2 | Method overview | 582 |
| 16.15.3 | TJSONNumber.JSONType | 582 |
| 16.15.4 | TJSONNumber.NumberType | 582 |
| 16.16 | TJSONObject | 582 |
| 16.16.1 | Description | 582 |
| 16.16.2 | Method overview | 583 |
| 16.16.3 | Property overview | 583 |
| 16.16.4 | TJSONObject.Create | 583 |
| 16.16.5 | TJSONObject.Destroy | 584 |</p>
<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>16.16.6</td>
<td>TJSONObject.JSONType</td>
<td>584</td>
</tr>
<tr>
<td>16.16.7</td>
<td>TJSONObject.Clone</td>
<td>584</td>
</tr>
<tr>
<td>16.16.8</td>
<td>TJSONObject.GetEnumerator</td>
<td>585</td>
</tr>
<tr>
<td>16.16.9</td>
<td>TJSONObject.Iterate</td>
<td>585</td>
</tr>
<tr>
<td>16.16.10</td>
<td>TJSONObject.IndexOf</td>
<td>585</td>
</tr>
<tr>
<td>16.16.11</td>
<td>TJSONObject.IndexOfName</td>
<td>585</td>
</tr>
<tr>
<td>16.16.12</td>
<td>TJSONObject.Find</td>
<td>586</td>
</tr>
<tr>
<td>16.16.13</td>
<td>TJSONObject.Get</td>
<td>586</td>
</tr>
<tr>
<td>16.16.14</td>
<td>TJSONObject.Clear</td>
<td>587</td>
</tr>
<tr>
<td>16.16.15</td>
<td>TJSONObject.Add</td>
<td>587</td>
</tr>
<tr>
<td>16.16.16</td>
<td>TJSONObject.Delete</td>
<td>587</td>
</tr>
<tr>
<td>16.16.17</td>
<td>TJSONObject.Remove</td>
<td>588</td>
</tr>
<tr>
<td>16.16.18</td>
<td>TJSONObject.Extract</td>
<td>588</td>
</tr>
<tr>
<td>16.16.19</td>
<td>TJSONObject.UnquotedMemberNames</td>
<td>588</td>
</tr>
<tr>
<td>16.16.20</td>
<td>TJSONObject.Names</td>
<td>589</td>
</tr>
<tr>
<td>16.16.21</td>
<td>TJSONObject.Elements</td>
<td>589</td>
</tr>
<tr>
<td>16.16.22</td>
<td>TJSONObject.Types</td>
<td>589</td>
</tr>
<tr>
<td>16.16.23</td>
<td>TJSONObject.Nulls</td>
<td>590</td>
</tr>
<tr>
<td>16.16.24</td>
<td>TJSONObject.Floats</td>
<td>590</td>
</tr>
<tr>
<td>16.16.25</td>
<td>TJSONObject.Integers</td>
<td>590</td>
</tr>
<tr>
<td>16.16.26</td>
<td>TJSONObject.Int64s</td>
<td>591</td>
</tr>
<tr>
<td>16.16.27</td>
<td>TJSONObject.QWords</td>
<td>591</td>
</tr>
<tr>
<td>16.16.28</td>
<td>TJSONObject.LargeInts</td>
<td>591</td>
</tr>
<tr>
<td>16.16.29</td>
<td>TJSONObject.UnicodeStrings</td>
<td>591</td>
</tr>
<tr>
<td>16.16.30</td>
<td>TJSONObject.Strings</td>
<td>592</td>
</tr>
<tr>
<td>16.16.31</td>
<td>TJSONObject.Booleans</td>
<td>592</td>
</tr>
<tr>
<td>16.16.32</td>
<td>TJSONObject.Arrays</td>
<td>593</td>
</tr>
<tr>
<td>16.16.33</td>
<td>TJSONObject.Objects</td>
<td>593</td>
</tr>
<tr>
<td>16.17</td>
<td>TJSONQWordNumber</td>
<td>593</td>
</tr>
<tr>
<td>16.17.1</td>
<td>Description</td>
<td>593</td>
</tr>
<tr>
<td>16.17.2</td>
<td>Method overview</td>
<td>593</td>
</tr>
<tr>
<td>16.17.3</td>
<td>TJSONQWordNumber.Create</td>
<td>593</td>
</tr>
<tr>
<td>16.17.4</td>
<td>TJSONQWordNumber.NumberType</td>
<td>593</td>
</tr>
<tr>
<td>16.17.5</td>
<td>TJSONQWordNumber.Clear</td>
<td>594</td>
</tr>
<tr>
<td>16.17.6</td>
<td>TJSONQWordNumber.Clone</td>
<td>594</td>
</tr>
<tr>
<td>16.18</td>
<td>TString</td>
<td>594</td>
</tr>
<tr>
<td>16.18.1</td>
<td>Description</td>
<td>594</td>
</tr>
<tr>
<td>16.18.2</td>
<td>Method overview</td>
<td>594</td>
</tr>
<tr>
<td>16.18.3</td>
<td>TString.Create</td>
<td>594</td>
</tr>
<tr>
<td>16.18.4</td>
<td>TString.JSONType</td>
<td>595</td>
</tr>
</tbody>
</table>
17 Reference for unit 'fpmimetypes'

17.1 Used units

17.2 Overview

17.3 Procedures and functions

17.4 TFPMimeType

17.5 TMimeType

18 Reference for unit 'ftimer'

51
19.8.3 TMOFile.Create ............................................. 611
19.8.4 TMOFile.Destroy ........................................... 611
19.8.5 TMOFile.Translate ......................................... 611

20 Reference for unit 'IBConnection' .................................. 613

20.1 Used units ..................................................... 613
20.2 Constants, types and variables .................................. 613
  20.2.1 Constants .................................................. 613
  20.2.2 Types ...................................................... 613
20.3 TDatabaseInfo .................................................. 614
20.4 EIBDatabaseError ............................................... 614
  20.4.1 Description ............................................... 614
  20.4.2 Property overview ...................................... 614
  20.4.3 EIBDatabaseError.StatusVector ......................... 614
  20.4.4 EIBDatabaseError.GDSErrorCode ......................... 614
20.5 TIBConnection ................................................ 615
  20.5.1 Description ............................................... 615
  20.5.2 Method overview ....................................... 616
  20.5.3 Property overview ...................................... 616
  20.5.4 TIBConnection.Create .................................... 616
  20.5.5 TIBConnection.GetConnectionInfo ....................... 616
  20.5.6 TIBConnection.CreateDB .................................. 617
  20.5.7 TIBConnection.DropDB ................................... 617
  20.5.8 TIBConnection.BlobSegmentSize ......................... 617
  20.5.9 TIBConnection.ODSMajorVersion ......................... 618
  20.5.10 TIBConnection.DatabaseName ......................... 618
  20.5.11 TIBConnection.Dialect .................................. 618
  20.5.12 TIBConnection.CheckTransactionParams .................. 619
  20.5.13 TIBConnection.KeepConnection ......................... 619
  20.5.14 TIBConnection.LoginPrompt ............................ 619
  20.5.15 TIBConnection.Params .................................. 619
  20.5.16 TIBConnection.OnLogin .................................. 620
  20.5.17 TIBConnection.Port .................................... 620
  20.5.18 TIBConnection.UseConnectionCharSetIfNone ............ 620
  20.5.19 TIBConnection.WireCompression ....................... 620
20.6 TIBConnectionDef ............................................. 621
  20.6.1 Description ............................................... 621
  20.6.2 Method overview ....................................... 621
  20.6.3 TIBConnectionDef.TypeName ............................ 621
  20.6.4 TIBConnectionDef.ConnectionClass ..................... 621
20.6.5 TIBConnectionDef.Description ........................................... 621
20.6.6 TIBConnectionDef.DefaultLibraryName .................................. 622
20.6.7 TIBConnectionDef.LoadFunction .......................................... 622
20.6.8 TIBConnectionDef.UnLoadFunction ...................................... 622
20.6.9 TIBConnectionDef.LoadedLibraryName .................................... 622
20.7 TIBCursor ............................................................................. 622
20.7.1 Description ......................................................................... 622
20.8 TIBTrans .............................................................................. 623
20.8.1 Description ......................................................................... 623

21 Reference for unit 'idea' .......................................................... 624
21.1 Used units ............................................................................. 624
21.2 Overview ............................................................................... 624
21.3 Constants, types and variables .............................................. 624
  21.3.1 Constants .......................................................................... 624
  21.3.2 Types ................................................................................ 625
21.4 Procedures and functions ...................................................... 625
  21.4.1 CipherIdea ......................................................................... 625
  21.4.2 DeKeyIdea ......................................................................... 625
  21.4.3 EnKeyIdea ......................................................................... 626
21.5 EIDEAError ........................................................................... 626
  21.5.1 Description ......................................................................... 626
21.6 TIDEADeCryptStream .............................................................. 626
  21.6.1 Description ......................................................................... 626
  21.6.2 Method overview ............................................................... 626
  21.6.3 TIDEADeCryptStream.Create ............................................ 627
  21.6.4 TIDEADeCryptStream.Read ................................................. 627
  21.6.5 TIDEADeCryptStream.Seek .................................................. 627
21.7 TIDEAEncryptStream ............................................................... 628
  21.7.1 Description ......................................................................... 628
  21.7.2 Method overview ............................................................... 628
  21.7.3 TIDEAEncryptStream.Create ............................................ 628
  21.7.4 TIDEAEncryptStream.Destroy .............................................. 628
  21.7.5 TIDEAEncryptStream.Write ............................................... 629
  21.7.6 TIDEAEncryptStream.Seek .................................................. 629
  21.7.7 TIDEAEncryptStream.Flush ................................................. 629
21.8 TIDEAStream ......................................................................... 629
  21.8.1 Description ......................................................................... 629
  21.8.2 Method overview ............................................................... 630
  21.8.3 Property overview ............................................................. 630
22 Reference for unit 'inicol' 631
   22.1 Used units 631
   22.2 Overview 631
   22.3 Constants, types and variables 631
     22.3.1 Constants 631
   22.4 EIniCol 632
     22.4.1 Description 632
   22.5 TIniCollection 632
     22.5.1 Description 632
     22.5.2 Method overview 632
     22.5.3 Property overview 632
     22.5.4 TIniCollection.Load 632
     22.5.5 TIniCollection.Save 633
     22.5.6 TIniCollection.SaveToIni 633
     22.5.7 TIniCollection.SaveToFile 633
     22.5.8 TIniCollection.LoadFromIni 634
     22.5.9 TIniCollection.LoadFromFile 634
     22.5.10 TIniCollection.Prefix 634
     22.5.11 TIniCollection.SectionPrefix 635
     22.5.12 TIniCollection.FileName 635
     22.5.13 TIniCollection.GlobalSection 635
   22.6 TIniCollectionItem 636
     22.6.1 Description 636
     22.6.2 Method overview 636
     22.6.3 Property overview 636
     22.6.4 TIniCollectionItem.SaveToIni 636
     22.6.5 TIniCollectionItem.LoadFromIni 636
     22.6.6 TIniCollectionItem.SaveToFile 637
     22.6.7 TIniCollectionItem.LoadFromFile 637
     22.6.8 TIniCollectionItem.SectionName 637
   22.7 TNamedIniCollection 638
     22.7.1 Description 638
     22.7.2 Method overview 638
     22.7.3 Property overview 638
     22.7.4 TNamedIniCollection.IndexOfUserData 638
     22.7.5 TNamedIniCollection.IndexOfName 638
     22.7.6 TNamedIniCollection.FindByName 639
22.7.7 TNamedIniCollection.FindByUserData .................................. 639
22.7.8 TNamedIniCollection.NamedItems ........................................ 639
22.8 TNamedIniCollectionItem ...................................................... 639
  22.8.1 Description ............................................................... 639
  22.8.2 Property overview ........................................................ 639
  22.8.3 TNamedIniCollectionItem.UserData ................................... 640
  22.8.4 TNamedIniCollectionItem.Name ......................................... 640
23 Reference for unit 'IniFiles' .................................................... 641
  23.1 Used units ................................................................. 641
  23.2 Overview ................................................................. 641
  23.3 Constants, types and variables ........................................... 641
    23.3.1 Types ................................................................. 641
  23.4 TCustomIniFile ............................................................. 643
    23.4.1 Description ........................................................... 643
    23.4.2 Method overview ...................................................... 643
    23.4.3 Property overview .................................................... 644
    23.4.4 TCustomIniFile.Create .............................................. 644
    23.4.5 TCustomIniFile.Destroy .............................................. 645
    23.4.6 TCustomIniFile.SetBoolStringValues .............................. 645
    23.4.7 TCustomIniFile.SectionExists ..................................... 645
    23.4.8 TCustomIniFile.ReadString ......................................... 645
    23.4.9 TCustomIniFile.WriteString ........................................ 646
    23.4.10 TCustomIniFile.ReadInteger ...................................... 646
    23.4.11 TCustomIniFile.WriteInteger ..................................... 646
    23.4.12 TCustomIniFile.ReadInt64 ......................................... 646
    23.4.13 TCustomIniFile.WriteInt64 ....................................... 647
    23.4.14 TCustomIniFile.ReadBool .......................................... 647
    23.4.15 TCustomIniFile.WriteBool ....................................... 647
    23.4.16 TCustomIniFile.ReadDate .......................................... 648
    23.4.17 TCustomIniFile.ReadDateTime ..................................... 648
    23.4.18 TCustomIniFile.ReadFloat ....................................... 648
    23.4.19 TCustomIniFile.ReadTime ........................................ 648
    23.4.20 TCustomIniFile.ReadBinaryStream ................................ 649
    23.4.21 TCustomIniFile.WriteDate ....................................... 649
    23.4.22 TCustomIniFile.WriteDateTime ................................... 649
    23.4.23 TCustomIniFile.WriteFloat ...................................... 650
    23.4.24 TCustomIniFile.WriteTime ....................................... 650
    23.4.25 TCustomIniFile.WriteBinaryStream ............................... 650
    23.4.26 TCustomIniFile.ReadSection ..................................... 650
23.7 TIniFileKey . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 661
   23.7.1 Description . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 661
   23.7.2 Method overview . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 661
   23.7.3 Property overview . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 661
   23.7.4 TIniFileKey.Create . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 661
   23.7.5 TIniFileKey.Ident . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 661
   23.7.6 TIniFileKey.Value . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 661
23.8 TIniFileKeyList . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 662
   23.8.1 Description . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 662
   23.8.2 Method overview . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 662
   23.8.3 Property overview . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 662
   23.8.4 TIniFileKeyList.Destroy . . . . . . . . . . . . . . . . . . . . . . . . . . . . 662
   23.8.5 TIniFileKeyList.Clear . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 662
   23.8.6 TIniFileKeyList.Items . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 662
23.9 TIniFileSection . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 663
   23.9.1 Description . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 663
   23.9.2 Method overview . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 663
   23.9.3 Property overview . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 663
   23.9.4 TIniFileSection.Empty . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 663
   23.9.5 TIniFileSection.Create . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 663
   23.9.6 TIniFileSection.Destroy . . . . . . . . . . . . . . . . . . . . . . . . . . . . 664
   23.9.7 TIniFileSection.Name . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 664
   23.9.8 TIniFileSection.KeyList . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 664
23.10 TIniFileSectionList . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 664
   23.10.1 Description . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 664
   23.10.2 Method overview . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 664
   23.10.3 Property overview . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 665
   23.10.4 TIniFileSectionList.Destroy . . . . . . . . . . . . . . . . . . . . . . . . . . 665
   23.10.5 TIniFileSectionList.Clear . . . . . . . . . . . . . . . . . . . . . . . . . . . 665
   23.10.6 TIniFileSectionList.Items . . . . . . . . . . . . . . . . . . . . . . . . . . . 665
23.11 TMemIniFile . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 665
   23.11.1 Description . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 665
   23.11.2 Method overview . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 666
   23.11.3 TMemIniFile.Create . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 666
   23.11.4 TMemIniFile.Clear . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 666
   23.11.5 TMemIniFile.GetString . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 666
   23.11.6 TMemIniFile.Rename . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 667
   23.11.7 TMemIniFile.SetStrings . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 667
23.12 TStringHash . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 667
   23.12.1 Description . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 667
### 24 Reference for unit 'iostream'

24.1 Used units ................................................. 670
24.2 Overview ................................................. 670
24.3 Constants, types and variables
   24.3.1 Types .............................................. 670
24.4 EIOStreamError ........................................... 671
   24.4.1 Description ....................................... 671
24.5 TIOStream ............................................... 671
   24.5.1 Description ....................................... 671
   24.5.2 Method overview ................................... 671
   24.5.3 TIOStream.Create .................................. 671
   24.5.4 TIOStream.Read .................................... 671
   24.5.5 TIOStream.Write ................................... 672
   24.5.6 TIOStream.Seek .................................... 672

### 25 Reference for unit 'libtar'

25.1 Used units ................................................. 673
25.2 Overview ................................................. 673
25.3 Constants, types and variables
   25.3.1 Constants ......................................... 673
   25.3.2 Types ............................................... 674
25.4 Procedures and functions
   25.4.1 ClearDirRec ....................................... 676
   25.4.2 ConvertFilename ................................... 676
   25.4.3 FileTimeGMT ...................................... 676
   25.4.4 PermissionString .................................. 676
25.5 TTarArchive .............................................. 677
   25.5.1 Description ....................................... 677
   25.5.2 Method overview ................................... 677
   25.5.3 TTarArchive.Create .................................. 677
25.5.4 TTarArchive.Destroy .................................................. 677
25.5.5 TTarArchive.Reset .................................................. 677
25.5.6 TTarArchive.FindNext ............................................. 678
25.5.7 TTarArchive.ReadFile ............................................... 678
25.5.8 TTarArchive.GetFilePos ............................................. 678
25.5.9 TTarArchive.SetFilePos ............................................. 679

25.6 TTarWriter ................................................................. 679
25.6.1 Description ......................................................... 679
25.6.2 Method overview .................................................. 679
25.6.3 Property overview ................................................ 679
25.6.4 TTarWriter.Create ................................................ 679
25.6.5 TTarWriter.Destroy ................................................ 680
25.6.6 TTarWriter.AddFile ............................................... 680
25.6.7 TTarWriter.AddStream ............................................ 680
25.6.8 TTarWriter.AddString ............................................ 681
25.6.9 TTarWriter.AddDir ............................................... 681
25.6.10 TTarWriter.AddSymbolicLink .................................. 681
25.6.11 TTarWriter.AddLink ............................................. 682
25.6.12 TTarWriter.AddVolumeHeader ................................ 682
25.6.13 TTarWriter.Finalize .............................................. 682
25.6.14 TTarWriter.Permissions .......................................... 682
25.6.15 TTarWriter.UID .................................................. 683
25.6.16 TTarWriter.GID .................................................. 683
25.6.17 TTarWriter.UserName ........................................... 683
25.6.18 TTarWriter.GroupName ........................................... 683
25.6.19 TTarWriter.Mode ................................................ 684
25.6.20 TTarWriter.Magic .............................................. 684

26 Reference for unit 'memds' .............................................. 685
26.1 Used units ............................................................... 685
26.2 Overview ................................................................. 685
26.3 Constants, types and variables .................................... 685
  26.3.1 Constants ........................................................ 685
26.4 MDSError ................................................................. 686
  26.4.1 Description ....................................................... 686
26.5 TMemDataset ............................................................ 687
  26.5.1 Description ....................................................... 687
  26.5.2 Method overview ................................................ 688
  26.5.3 Property overview ............................................... 689
  26.5.4 TMemDataset.Create ............................................ 689
26.5.5 TMemDataset.Destroy ........................................... 689
26.5.6 TMemDataset.BookmarkValid ................................... 690
26.5.7 TMemDataset.CompareBookmarks ................................. 690
26.5.8 TMemDataset.CreateBlobStream ................................. 690
26.5.9 TMemDataset.Locate ............................................. 691
26.5.10 TMemDataset.Lookup ........................................... 691
26.5.11 TMemDataset.CreateTable .......................... 691
26.5.12 TMemDataset.DataSize ........................................ 692
26.5.13 TMemDataset.Clear ............................................. 692
26.5.14 TMemDataset.SaveToFile ................................. 692
26.5.15 TMemDataset.SaveToStream ................................. 693
26.5.16 TMemDataset.LoadFromStream ................................. 693
26.5.17 TMemDataset.LoadFromFile ................................. 693
26.5.18 TMemDataset.CopyFromDataset ................................. 694
26.5.19 TMemDataset.FileModified .................................... 695
26.5.20 TMemDataset.Filter ........................................... 695
26.5.21 TMemDataset.FileName ........................................ 695
26.5.22 TMemDataset.Filtered ......................................... 696
26.5.23 TMemDataset.Active ........................................... 696
26.5.24 TMemDataset.FieldDefs ....................................... 696
26.5.25 TMemDataset.BeforeOpen ...................................... 696
26.5.26 TMemDataset.AfterOpen ...................................... 696
26.5.27 TMemDataset.BeforeClose .................................... 697
26.5.28 TMemDataset.AfterClose ..................................... 697
26.5.29 TMemDataset.BeforeInsert .................................. 697
26.5.30 TMemDataset.AfterInsert .................................... 697
26.5.31 TMemDataset.BeforeEdit ...................................... 697
26.5.32 TMemDataset.AfterEdit ...................................... 697
26.5.33 TMemDataset.BeforePost ..................................... 698
26.5.34 TMemDataset.AfterPost ....................................... 698
26.5.35 TMemDataset.BeforeCancel .................................. 698
26.5.36 TMemDataset.AfterCancel .................................... 698
26.5.37 TMemDataset.BeforeDelete .................................. 698
26.5.38 TMemDataset.AfterDelete .................................... 698
26.5.39 TMemDataset.BeforeScroll .................................. 699
26.5.40 TMemDataset.AfterScroll .................................... 699
26.5.41 TMemDataset.OnDeleteError .................................. 699
26.5.42 TMemDataset.OnEditError .................................... 699
26.5.43 TMemDataset.OnNewRecord ................................... 699
26.5.44 TMemDataset.OnPostError .................................... 700
## Reference for unit 'MSSQLConn'

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>27.1 Used units</td>
<td>701</td>
</tr>
<tr>
<td>27.2 Overview</td>
<td>701</td>
</tr>
<tr>
<td>27.3 Constants, types and variables</td>
<td>701</td>
</tr>
<tr>
<td>27.3.1 Variables</td>
<td>701</td>
</tr>
<tr>
<td>27.4 EMSSQLDatabaseError</td>
<td>701</td>
</tr>
<tr>
<td>27.4.1 Description</td>
<td>701</td>
</tr>
<tr>
<td>27.4.2 Property overview</td>
<td>702</td>
</tr>
<tr>
<td>27.4.3 EMSSQLDatabaseError.DBErrorCode</td>
<td>702</td>
</tr>
<tr>
<td>27.5 TMSSQLConnection</td>
<td>702</td>
</tr>
<tr>
<td>27.5.1 Description</td>
<td>702</td>
</tr>
<tr>
<td>27.5.2 Method overview</td>
<td>703</td>
</tr>
<tr>
<td>27.5.3 Property overview</td>
<td>703</td>
</tr>
<tr>
<td>27.5.4 TMSSQLConnection.Create</td>
<td>703</td>
</tr>
<tr>
<td>27.5.5 TMSSQLConnection.GetConnectionInfo</td>
<td>703</td>
</tr>
<tr>
<td>27.5.6 TMSSQLConnection.CreateDB</td>
<td>703</td>
</tr>
<tr>
<td>27.5.7 TMSSQLConnection.DropDB</td>
<td>704</td>
</tr>
<tr>
<td>27.5.8 TMSSQLConnection.Password</td>
<td>704</td>
</tr>
<tr>
<td>27.5.9 TMSSQLConnection.Transaction</td>
<td>704</td>
</tr>
<tr>
<td>27.5.10 TMSSQLConnection.UserName</td>
<td>704</td>
</tr>
<tr>
<td>27.5.11 TMSSQLConnection.CharSet</td>
<td>705</td>
</tr>
<tr>
<td>27.5.12 TMSSQLConnection.HostName</td>
<td>705</td>
</tr>
<tr>
<td>27.5.13 TMSSQLConnection.Connected</td>
<td>705</td>
</tr>
<tr>
<td>27.5.14 TMSSQLConnection.Role</td>
<td>705</td>
</tr>
<tr>
<td>27.5.15 TMSSQLConnection.DatabaseName</td>
<td>706</td>
</tr>
<tr>
<td>27.5.16 TMSSQLConnection.KeepConnection</td>
<td>706</td>
</tr>
<tr>
<td>27.5.17 TMSSQLConnection.LoginPrompt</td>
<td>706</td>
</tr>
<tr>
<td>27.5.18 TMSSQLConnection.Params</td>
<td>706</td>
</tr>
<tr>
<td>27.5.19 TMSSQLConnection.OnLogin</td>
<td>707</td>
</tr>
<tr>
<td>27.6 TMSSQLConnectionDef</td>
<td>707</td>
</tr>
<tr>
<td>27.6.1 Description</td>
<td>707</td>
</tr>
<tr>
<td>27.6.2 Method overview</td>
<td>707</td>
</tr>
<tr>
<td>27.6.3 TMSSQLConnectionDef.TypeName</td>
<td>707</td>
</tr>
<tr>
<td>27.6.4 TMSSQLConnectionDef.ConnectionClass</td>
<td>707</td>
</tr>
<tr>
<td>27.6.5 TMSSQLConnectionDef.Description</td>
<td>708</td>
</tr>
<tr>
<td>27.6.6 TMSSQLConnectionDef.DefaultLibraryName</td>
<td>708</td>
</tr>
<tr>
<td>27.6.7 TMSSQLConnectionDef.LoadFunction</td>
<td>708</td>
</tr>
<tr>
<td>27.6.8 TMSSQLConnectionDef.UnLoadFunction</td>
<td>708</td>
</tr>
<tr>
<td>Section</td>
<td>Description</td>
</tr>
<tr>
<td>---------</td>
<td>-------------</td>
</tr>
<tr>
<td>27.6.9</td>
<td>TMSSQLConnectionDef.LoadedLibraryName</td>
</tr>
<tr>
<td>27.7</td>
<td>TSybaseConnection</td>
</tr>
<tr>
<td>27.7.1</td>
<td>Description</td>
</tr>
<tr>
<td>27.7.2</td>
<td>Method overview</td>
</tr>
<tr>
<td>27.7.3</td>
<td>TSybaseConnection.Create</td>
</tr>
<tr>
<td>27.8</td>
<td>TSybaseConnectionDef</td>
</tr>
<tr>
<td>27.8.1</td>
<td>Description</td>
</tr>
<tr>
<td>27.8.2</td>
<td>Method overview</td>
</tr>
<tr>
<td>27.8.3</td>
<td>TSybaseConnectionDef.TypeName</td>
</tr>
<tr>
<td>27.8.4</td>
<td>TSybaseConnectionDef.ConnectionClass</td>
</tr>
<tr>
<td>27.8.5</td>
<td>TSybaseConnectionDef.Description</td>
</tr>
<tr>
<td>28</td>
<td>Reference for unit 'nullstream'</td>
</tr>
<tr>
<td>28.1</td>
<td>Used units</td>
</tr>
<tr>
<td>28.2</td>
<td>Overview</td>
</tr>
<tr>
<td>28.3</td>
<td>ENullStreamError</td>
</tr>
<tr>
<td>28.3.1</td>
<td>Description</td>
</tr>
<tr>
<td>28.4</td>
<td>TNullStream</td>
</tr>
<tr>
<td>28.4.1</td>
<td>Description</td>
</tr>
<tr>
<td>28.4.2</td>
<td>Method overview</td>
</tr>
<tr>
<td>28.4.3</td>
<td>TNullStream.Read</td>
</tr>
<tr>
<td>28.4.4</td>
<td>TNullStream.Write</td>
</tr>
<tr>
<td>28.4.5</td>
<td>TNullStream.Seek</td>
</tr>
<tr>
<td>28.4.6</td>
<td>TNullStream.Create</td>
</tr>
<tr>
<td>29</td>
<td>Reference for unit 'Pipes'</td>
</tr>
<tr>
<td>29.1</td>
<td>Used units</td>
</tr>
<tr>
<td>29.2</td>
<td>Overview</td>
</tr>
<tr>
<td>29.3</td>
<td>Constants, types and variables</td>
</tr>
<tr>
<td>29.3.1</td>
<td>Constants</td>
</tr>
<tr>
<td>29.4</td>
<td>Procedures and functions</td>
</tr>
<tr>
<td>29.4.1</td>
<td>CreatePipeHandles</td>
</tr>
<tr>
<td>29.4.2</td>
<td>CreatePipeStreams</td>
</tr>
<tr>
<td>29.5</td>
<td>EPipeCreation</td>
</tr>
<tr>
<td>29.5.1</td>
<td>Description</td>
</tr>
<tr>
<td>29.6</td>
<td>EPipeError</td>
</tr>
<tr>
<td>29.6.1</td>
<td>Description</td>
</tr>
<tr>
<td>29.7</td>
<td>EPipeSeek</td>
</tr>
<tr>
<td>29.7.1</td>
<td>Description</td>
</tr>
<tr>
<td>29.8</td>
<td>TInputPipeStream</td>
</tr>
<tr>
<td>29.8.1</td>
<td>Description</td>
</tr>
</tbody>
</table>
30 Reference for unit ‘pooledmm’

30.1 Used units ................................................. 719
30.2 Overview .................................................. 719
30.3 Constants, types and variables .......................... 719
  30.3.1 Types ................................................. 719
30.4 TPooledMemManagerItem ................................. 719
30.5 TNonFreePooledMemManager ............................... 720
  30.5.1 Description ......................................... 720
  30.5.2 Method overview ...................................... 720
  30.5.3 Property overview .................................... 720
  30.5.4 TNonFreePooledMemManager.Clear ................. 720
  30.5.5 TNonFreePooledMemManager.Create ................ 720
  30.5.6 TNonFreePooledMemManager.Destroy ................ 721
  30.5.7 TNonFreePooledMemManagerNewItem ............... 721
  30.5.8 TNonFreePooledMemManager.EnumerateItems ........ 721
  30.5.9 TNonFreePooledMemManager.ItemSize ............. 721
30.6 TPooledMemManager ....................................... 722
  30.6.1 Description ......................................... 722
  30.6.2 Method overview ...................................... 722
  30.6.3 Property overview .................................... 722
  30.6.4 TPooledMemManager.Clear ......................... 722
  30.6.5 TPooledMemManager.Create ....................... 722
  30.6.6 TPooledMemManager.Destroy ....................... 723
  30.6.7 TPooledMemManager.MinimumFreeCount ............ 723
  30.6.8 TPooledMemManager.MaximumFreeCountRatio ........ 723
  30.6.9 TPooledMemManager.Count ......................... 723
30.6.10 TPooledMemManager.FreeCount . . . . . . . . . . . . . . . . . . . . . . . 724
30.6.11 TPooledMemManager.AllocatedCount . . . . . . . . . . . . . . . . . . . 724
30.6.12 TPooledMemManager.FreedCount . . . . . . . . . . . . . . . . . . . . . . . 724

31 Reference for unit 'process' 725
31.1 Used units . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 725
31.2 Overview . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 725
31.3 Constants, types and variables . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 725
31.3.1 Types . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 725
31.3.2 Variables . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 729
31.4 Procedures and functions . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 729
31.4.1 CommandToList . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 729
31.4.2 DetectXTerm . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 729
31.4.3 RunCommand . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 730
31.4.4 RunCommandIndir . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 730
31.5 EProcess . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 731
31.5.1 Description . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 731
31.6 TPROCESS . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 731
31.6.1 Description . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 731
31.6.2 Method overview . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 731
31.6.3 Property overview . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 732
31.6.4 TPROCESS.Create . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 733
31.6.5 TPROCESS.Destroy . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 733
31.6.6 TPROCESS.Execute . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 733
31.6.7 TPROCESS.CloseInput . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 734
31.6.8 TPROCESS.CloseOutput . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 734
31.6.9 TPROCESS.CloseStderr . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 734
31.6.10 TPROCESS.Resume . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 734
31.6.11 TPROCESS.Suspend . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 735
31.6.12 TPROCESS.Terminate . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 735
31.6.13 TPROCESS.WaitOnExit . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 735
31.6.14 TPROCESS.ReadInputStream . . . . . . . . . . . . . . . . . . . . . . . . . . 736
31.6.15 TPROCESS.RunCommandLoop . . . . . . . . . . . . . . . . . . . . . . . . . . 736
31.6.16 TPROCESS.WindowRect . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 736
31.6.17 TPROCESS.Handle . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 737
31.6.18 TPROCESS.ProcessHandle . . . . . . . . . . . . . . . . . . . . . . . . . . . 737
31.6.19 TPROCESS.ThreadHandle . . . . . . . . . . . . . . . . . . . . . . . . . . . . 737
31.6.20 TPROCESS.ProcessID . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 737
31.6.21 TPROCESS.ThreadID . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 738
31.6.22 TPROCESS.Input . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 738
31.6.23 TPROCESS.Output .............................................. 738
31.6.24 TPROCESS.Stderr ............................................ 739
31.6.25 TPROCESS.ExitStatus ........................................ 739
31.6.26 TPROCESS.ExitCode .......................................... 740
31.6.27 TPROCESS.InheritHandles ................................. 740
31.6.28 TPROCESS.OnRunCommandEvent ............................ 740
31.6.29 TPROCESS.RunCommandSleepTime ......................... 740
31.6.30 TPROCESS.OnForkEvent ..................................... 741
31.6.31 TPROCESS.PipeBufferSize ................................. 741
31.6.32 TPROCESS.Active ........................................... 741
31.6.33 TPROCESS.ApplicationName ............................... 741
31.6.34 TPROCESS.CommandLine .................................. 742
31.6.35 TPROCESS.Executable ...................................... 742
31.6.36 TPROCESS.Parameters ...................................... 743
31.6.37 TPROCESS.ConsoleTitle ................................... 743
31.6.38 TPROCESS.CurrentDirectory ............................. 744
31.6.39 TPROCESS.Desktop .......................................... 744
31.6.40 TPROCESS.Environment ..................................... 744
31.6.41 TPROCESS.Options ......................................... 745
31.6.42 TPROCESS.Priority ......................................... 745
31.6.43 TPROCESS.StartupOptions ................................. 746
31.6.44 TPROCESS.Running .......................................... 746
31.6.45 TPROCESS.ShowWindow .................................... 747
31.6.46 TPROCESS.WindowColumns ................................ 747
31.6.47 TPROCESS.WindowHeight .................................. 747
31.6.48 TPROCESS.WindowLeft ..................................... 748
31.6.49 TPROCESS.WindowRows ..................................... 748
31.6.50 TPROCESS.WindowTop ...................................... 748
31.6.51 TPROCESS.WindowWidth ................................... 749
31.6.52 TPROCESS.FillAttribute ................................ 749
31.6.53 TPROCESS.XTermProgram .................................. 749

32 Reference for unit 'RttiUtils' 750
32.1 Used units ......................................................... 750
32.2 Overview .......................................................... 750
32.3 Constants, types and variables ............................... 750
  32.3.1 Constants ....................................................... 750
  32.3.2 Types ........................................................... 750
  32.3.3 Variables ....................................................... 751
32.4 Procedures and functions ..................................... 752
33.3.4 Variables ........................................ 763
33.4 TMsgHeader ........................................ 764
33.5 EIPCError ........................................ 764
  33.5.1 Description .................................. 764
33.6 TIPCClientComm .................................... 764
  33.6.1 Description .................................. 764
  33.6.2 Method overview .............................. 765
  33.6.3 Property overview ............................ 765
  33.6.4 TIPCClientComm.Create .................... 765
  33.6.5 TIPCClientComm.Connect ................... 765
  33.6.6 TIPCClientComm.Disconnect ............... 765
  33.6.7 TIPCClientComm.ServerRunning ............ 766
  33.6.8 TIPCClientComm.SendMessage .............. 766
  33.6.9 TIPCClientComm.Owner ..................... 766
33.7 TIPCServerComm ................................... 767
  33.7.1 Description .................................. 767
  33.7.2 Method overview .............................. 767
  33.7.3 Property overview ............................ 767
  33.7.4 TIPCServerComm.Create .................... 767
  33.7.5 TIPCServerComm.StartServer ............... 767
  33.7.6 TIPCServerComm.StopServer ................. 768
  33.7.7 TIPCServerComm.PeekMessage ............... 768
  33.7.8 TIPCServerComm.ReadMessage ............... 768
  33.7.9 TIPCServerComm.Owner ..................... 769
  33.7.10 TIPCServerComm.InstanceID ............... 769
33.8 TIPCServerMsg ..................................... 769
  33.8.1 Description .................................. 769
  33.8.2 Method overview .............................. 769
  33.8.3 Property overview ............................ 769
  33.8.4 TIPCServerMsg.Create .................... 770
  33.8.5 TIPCServerMsg.Destroy .................... 770
  33.8.6 TIPCServerMsg.Stream ..................... 770
  33.8.7 TIPCServerMsg.MsgType ..................... 770
  33.8.8 TIPCServerMsg.OwnsStream ................. 771
  33.8.9 TIPCServerMsg.StringMessage .............. 771
33.9 TIPCServerMsgQueue ............................... 771
  33.9.1 Description .................................. 771
  33.9.2 Method overview .............................. 771
  33.9.3 Property overview ............................ 771
  33.9.4 TIPCServerMsgQueue.Create ............... 772
34.12.4 TConnectionDef.ConnectionClass ........................................ 803
34.12.5 TConnectionDef.Description ............................................. 803
34.12.6 TConnectionDef.DefaultLibraryName .................................... 803
34.12.7 TConnectionDef.LoadFunction ........................................... 803
34.12.8 TConnectionDef.UnLoadFunction ........................................ 804
34.12.9 TConnectionDef.LoadedLibraryName ..................................... 804
34.12.10 TConnectionDef.ApplyParams ......................................... 804

34.13 TCustomSQLQuery ............................................................. 804
34.13.1 Description ................................................................. 804
34.13.2 Method overview .......................................................... 805
34.13.3 Property overview ........................................................ 805
34.13.4 TCustomSQLQuery.Create .............................................. 805
34.13.5 TCustomSQLQuery.Destroy .............................................. 805
34.13.6 TCustomSQLQuery.Prepare .............................................. 806
34.13.7 TCustomSQLQuery.UnPrepare ......................................... 806
34.13.8 TCustomSQLQuery.ExecSQL ............................................ 806
34.13.9 TCustomSQLQuery.SetSchemaInfo ..................................... 807
34.13.10 TCustomSQLQuery.RowsAffected ..................................... 807
34.13.11 TCustomSQLQuery.ParamByName ..................................... 807
34.13.12 TCustomSQLQuery.MacroByName ..................................... 808
34.13.13 TCustomSQLQuery.ApplyUpdates ..................................... 808
34.13.14 TCustomSQLQuery.Post ............................................... 808
34.13.15 TCustomSQLQuery.Delete ............................................. 808
34.13.16 TCustomSQLQuery.Prepared ......................................... 809
34.13.17 TCustomSQLQuery.SQLConnection .................................. 809
34.13.18 TCustomSQLQuery.SQLTransaction .................................. 809

34.14 TCustomSQLStatement ...................................................... 809
34.14.1 Description ................................................................. 809
34.14.2 Method overview .......................................................... 810
34.14.3 Property overview ........................................................ 810
34.14.4 TCustomSQLStatement.Create ....................................... 810
34.14.5 TCustomSQLStatement.Destroy ....................................... 810
34.14.6 TCustomSQLStatement.Prepare ....................................... 811
34.14.7 TCustomSQLStatement.Execute ....................................... 811
34.14.8 TCustomSQLStatement.Unprepare .................................... 811
34.14.9 TCustomSQLStatement.ParamByName ................................ 812
34.14.10 TCustomSQLStatement.RowsAffected ................................ 812
34.14.11 TCustomSQLStatement.Prepared .................................... 812

34.15 TServerIndexDefs ........................................................... 812
34.15.1 Description ................................................................. 812
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>34.15.2 Method overview</td>
<td>812</td>
</tr>
<tr>
<td>34.15.3 TServerIndexDefs.Create</td>
<td>813</td>
</tr>
<tr>
<td>34.15.4 TServerIndexDefs.Update</td>
<td>813</td>
</tr>
<tr>
<td>34.16 TSQLConnection</td>
<td>813</td>
</tr>
<tr>
<td>34.16.1 Description</td>
<td>813</td>
</tr>
<tr>
<td>34.16.2 Method overview</td>
<td>813</td>
</tr>
<tr>
<td>34.16.3 Property overview</td>
<td>814</td>
</tr>
<tr>
<td>34.16.4 TSQLConnection.Create</td>
<td>814</td>
</tr>
<tr>
<td>34.16.5 TSQLConnection.Destroy</td>
<td>814</td>
</tr>
<tr>
<td>34.16.6 TSQLConnection.StartTransaction</td>
<td>815</td>
</tr>
<tr>
<td>34.16.7 TSQLConnection.EndTransaction</td>
<td>815</td>
</tr>
<tr>
<td>34.16.8 TSQLConnection.ExecuteDirect</td>
<td>815</td>
</tr>
<tr>
<td>34.16.9 TSQLConnection.GetObjectNames</td>
<td>816</td>
</tr>
<tr>
<td>34.16.10 TSQLConnection.GetTableNames</td>
<td>816</td>
</tr>
<tr>
<td>34.16.11 TSQLConnection.GetProcedureNames</td>
<td>816</td>
</tr>
<tr>
<td>34.16.12 TSQLConnection.GetFieldNames</td>
<td>816</td>
</tr>
<tr>
<td>34.16.13 TSQLConnection.GetSchemaNames</td>
<td>817</td>
</tr>
<tr>
<td>34.16.14 TSQLConnection.GetSequenceNames</td>
<td>817</td>
</tr>
<tr>
<td>34.16.15 TSQLConnection.GetConnectionInfo</td>
<td>817</td>
</tr>
<tr>
<td>34.16.16 TSQLConnection.GetStatementInfo</td>
<td>817</td>
</tr>
<tr>
<td>34.16.17 TSQLConnection.CreateDB</td>
<td>818</td>
</tr>
<tr>
<td>34.16.18 TSQLConnection.DropDB</td>
<td>818</td>
</tr>
<tr>
<td>34.16.19 TSQLConnection.GetNextValue</td>
<td>818</td>
</tr>
<tr>
<td>34.16.20 TSQLConnection.ConnOptions</td>
<td>819</td>
</tr>
<tr>
<td>34.16.21 TSQLConnection.Handle</td>
<td>819</td>
</tr>
<tr>
<td>34.16.22 TSQLConnection.FieldNameQuoteChars</td>
<td>819</td>
</tr>
<tr>
<td>34.16.23 TSQLConnection.Password</td>
<td>819</td>
</tr>
<tr>
<td>34.16.24 TSQLConnection.Transaction</td>
<td>820</td>
</tr>
<tr>
<td>34.16.25 TSQLConnection.UserName</td>
<td>820</td>
</tr>
<tr>
<td>34.16.26 TSQLConnection.CharSet</td>
<td>820</td>
</tr>
<tr>
<td>34.16.27 TSQLConnection.HostName</td>
<td>821</td>
</tr>
<tr>
<td>34.16.28 TSQLConnection.OnLog</td>
<td>821</td>
</tr>
<tr>
<td>34.16.29 TSQLConnection.LogEvents</td>
<td>821</td>
</tr>
<tr>
<td>34.16.30 TSQLConnection.Options</td>
<td>822</td>
</tr>
<tr>
<td>34.16.31 TSQLConnection.Role</td>
<td>822</td>
</tr>
<tr>
<td>34.16.32 TSQLConnection.Connected</td>
<td>822</td>
</tr>
<tr>
<td>34.16.33 TSQLConnection.DatabaseName</td>
<td>823</td>
</tr>
<tr>
<td>34.16.34 TSQLConnection.KeepConnection</td>
<td>823</td>
</tr>
<tr>
<td>34.16.35 TSQLConnection.LoginPrompt</td>
<td>823</td>
</tr>
<tr>
<td>34.16.36 TSQLConnection.Params</td>
<td>824</td>
</tr>
</tbody>
</table>
34.16.3 TSQLConnection.OnLogin ......................................................... 824
34.17 TSQLConnector ........................................................................ 824
  34.17.1 Description ........................................................................ 824
  34.17.2 Property overview ............................................................. 824
  34.17.3 TSQLConnector.ConnectorType .......................................... 825
34.18 TSQLCursor ............................................................................ 825
  34.18.1 Description ........................................................................ 825
34.19 TSQLDBFieldDef .................................................................... 825
  34.19.1 Description ........................................................................ 825
  34.19.2 Property overview ............................................................. 825
  34.19.3 TSQLDBFieldDef.SQLDBData ........................................... 825
34.20 TSQLDBFieldDefs ................................................................. 826
  34.20.1 Description ........................................................................ 826
34.21 TSQLDBParam ....................................................................... 826
  34.21.1 Description ........................................................................ 826
  34.21.2 Property overview ............................................................. 826
  34.21.3 TSQLDBParam.FieldDef .................................................... 826
  34.21.4 TSQLDBParam.SQLDBData .............................................. 826
34.22 TSQLDBParams ................................................................. 827
  34.22.1 Description ........................................................................ 827
34.23 TSQLHandle .......................................................................... 827
  34.23.1 Description ........................................................................ 827
34.24 TSQLQuery ........................................................................... 827
  34.24.1 Description ........................................................................ 827
  34.24.2 Property overview ............................................................. 829
  34.24.3 TSQLQuery.SchemaType .................................................... 830
  34.24.4 TSQLQuery.StatementType ............................................... 830
  34.24.5 TSQLQuery.MaxIndexesCount ......................................... 830
  34.24.6 TSQLQuery.FieldDefs ....................................................... 830
  34.24.7 TSQLQuery.Active ............................................................ 831
  34.24.8 TSQLQuery.AutoCalcFields .............................................. 831
  34.24.9 TSQLQuery.Filter .............................................................. 831
  34.24.10 TSQLQuery.Filtered ......................................................... 831
  34.24.11 TSQLQuery.AfterCancel .................................................. 831
  34.24.12 TSQLQuery.AfterClose .................................................... 831
  34.24.13 TSQLQuery.AfterDelete ................................................... 831
  34.24.14 TSQLQuery.AfterEdit ...................................................... 832
  34.24.15 TSQLQuery.AfterInsert ................................................... 832
  34.24.16 TSQLQuery.AfterOpen .................................................... 832
  34.24.17 TSQLQuery.AfterPost ...................................................... 832
34.25 TSQLScript ......................................................... 843
   34.25.1 Description .............................................. 843
   34.25.2 Method overview ........................................ 843
   34.25.3 Property overview ...................................... 843
   34.25.4 TSQLScript.Create ..................................... 843
   34.25.5 TSQLScript.Destroy .................................... 844
   34.25.6 TSQLScript.Execute .................................... 844
   34.25.7 TSQLScript.ExecuteScript .............................. 844
   34.25.8 TSQLScript.Aborted ..................................... 844
   34.25.9 TSQLScript.Line ........................................ 845
   34.25.10 TSQLScript.DataBase ................................. 845
   34.25.11 TSQLScript.Transaction .............................. 845
   34.25.12 TSQLScript.OnDirective .............................. 845
   34.25.13 TSQLScript.AutoCommit .............................. 846
   34.25.14 TSQLScript.UseDollarString ......................... 846
   34.25.15 TSQLScript.DollarStrings .......................... 846
   34.25.16 TSQLScript.Directives .............................. 847
   34.25.17 TSQLScript.Defines ................................. 847
   34.25.18 TSQLScript.Script ................................. 847
   34.25.19 TSQLScript.Terminator .............................. 848
   34.25.20 TSQLScript.CommentsinSQL .......................... 848
   34.25.21 TSQLScript.UseSetTerm .............................. 848
   34.25.22 TSQLScript.UseCommit .............................. 849
   34.25.23 TSQLScript.UseDefines .............................. 849
   34.25.24 TSQLScript.OnException ............................ 850
34.26 TSQLSequence .................................................. 850
   34.26.1 Description ........................................... 850
   34.26.2 Method overview ..................................... 850
   34.26.3 Property overview ................................... 850
   34.26.4 TSQLSequence.Create ................................ 850
   34.26.5 TSQLSequence.Assign ................................. 851
   34.26.6 TSQLSequence.Apply .................................. 851
   34.26.7 TSQLSequence.GetNextValue .......................... 851
   34.26.8 TSQLSequence.FieldName ............................. 851
   34.26.9 TSQLSequence.SequenceName ......................... 852
   34.26.10 TSQLSequence.IncrementBy ......................... 852
   34.26.11 TSQLSequence.ApplyEvent .......................... 852
34.27 TSQLStatement .................................................. 852
   34.27.1 Description ........................................... 852
   34.27.2 Property overview ................................... 853
35.4.1 Description .................................................. 864
35.4.2 Method overview .......................................... 864
35.4.3 Property overview .......................................... 864
35.4.4 TSqlObjectIdentifierList.AddIdentifier ............... 864
35.4.5 TSqlObjectIdentifierList.Identifiers ................. 865

36 Reference for unit 'streamcoll' .................................. 866
36.1 Used units ..................................................... 866
36.2 Overview ...................................................... 866
36.3 Procedures and functions .................................... 866
  36.3.1 ColReadBoolean ........................................... 866
  36.3.2 ColReadCurrency .......................................... 867
  36.3.3 ColReadDateTime .......................................... 867
  36.3.4 ColReadFloat ............................................. 867
  36.3.5 ColReadInteger ........................................... 867
  36.3.6 ColReadString ........................................... 868
  36.3.7 ColWriteBoolean .......................................... 868
  36.3.8 ColWriteCurrency ......................................... 868
  36.3.9 ColWriteDateTime ......................................... 868
  36.3.10 ColWriteFloat ........................................... 869
  36.3.11 ColWriteInteger ......................................... 869
  36.3.12 ColWriteString .......................................... 869
36.4 EStreamColl .................................................. 869
  36.4.1 Description ............................................... 869
36.5 TStreamCollection ........................................... 869
  36.5.1 Description ............................................... 869
  36.5.2 Method overview ......................................... 870
  36.5.3 Property overview ....................................... 870
  36.5.4 TStreamCollection.LoadFromStream ....................... 870
  36.5.5 TStreamCollection.SaveToStream ......................... 870
  36.5.6 TStreamCollection.Streaming ............................. 870
36.6 TStreamCollectionItem ....................................... 871
  36.6.1 Description ............................................... 871

37 Reference for unit 'streamex' .................................... 872
37.1 Used units .................................................... 872
37.2 Overview ..................................................... 872
37.3 Constants, types and variables .............................. 872
  37.3.1 Constants ................................................ 872
37.4 TBidirBinaryObjectReader .................................... 873
  37.4.1 Description ............................................... 873
<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>37.4.2</td>
<td>Property overview</td>
</tr>
<tr>
<td>37.4.3</td>
<td>TBidirBinaryObjectReader.Position</td>
</tr>
<tr>
<td>37.5</td>
<td>TBidirBinaryObjectWriter</td>
</tr>
<tr>
<td>37.5.1</td>
<td>Description</td>
</tr>
<tr>
<td>37.5.2</td>
<td>Property overview</td>
</tr>
<tr>
<td>37.5.3</td>
<td>TBidirBinaryObjectWriter.Position</td>
</tr>
<tr>
<td>37.6</td>
<td>TDelphiReader</td>
</tr>
<tr>
<td>37.6.1</td>
<td>Description</td>
</tr>
<tr>
<td>37.6.2</td>
<td>Method overview</td>
</tr>
<tr>
<td>37.6.3</td>
<td>Property overview</td>
</tr>
<tr>
<td>37.6.4</td>
<td>TDelphiReader.GetDriver</td>
</tr>
<tr>
<td>37.6.5</td>
<td>TDelphiReader.ReadStr</td>
</tr>
<tr>
<td>37.6.6</td>
<td>TDelphiReader.Read</td>
</tr>
<tr>
<td>37.6.7</td>
<td>TDelphiReader.Position</td>
</tr>
<tr>
<td>37.7</td>
<td>TDelphiWriter</td>
</tr>
<tr>
<td>37.7.1</td>
<td>Description</td>
</tr>
<tr>
<td>37.7.2</td>
<td>Method overview</td>
</tr>
<tr>
<td>37.7.3</td>
<td>Property overview</td>
</tr>
<tr>
<td>37.7.4</td>
<td>TDelphiWriter.GetDriver</td>
</tr>
<tr>
<td>37.7.5</td>
<td>TDelphiWriter.FlushBuffer</td>
</tr>
<tr>
<td>37.7.6</td>
<td>TDelphiWriter.Write</td>
</tr>
<tr>
<td>37.7.7</td>
<td>TDelphiWriter.WriteStr</td>
</tr>
<tr>
<td>37.7.8</td>
<td>TDelphiWriter.WriteValue</td>
</tr>
<tr>
<td>37.7.9</td>
<td>TDelphiWriter.Position</td>
</tr>
<tr>
<td>37.8</td>
<td>TFileReader</td>
</tr>
<tr>
<td>37.8.1</td>
<td>Description</td>
</tr>
<tr>
<td>37.8.2</td>
<td>Method overview</td>
</tr>
<tr>
<td>37.8.3</td>
<td>TFileReader.Create</td>
</tr>
<tr>
<td>37.8.4</td>
<td>TFileReader.Destroy</td>
</tr>
<tr>
<td>37.8.5</td>
<td>TFileReader.Reset</td>
</tr>
<tr>
<td>37.8.6</td>
<td>TFileReader.Close</td>
</tr>
<tr>
<td>37.8.7</td>
<td>TFileReader.ReadLine</td>
</tr>
<tr>
<td>37.9</td>
<td>TStreamHelper</td>
</tr>
<tr>
<td>37.9.1</td>
<td>Description</td>
</tr>
<tr>
<td>37.9.2</td>
<td>Method overview</td>
</tr>
<tr>
<td>37.9.3</td>
<td>TStreamHelper.ReadWordLE</td>
</tr>
<tr>
<td>37.9.4</td>
<td>TStreamHelper.ReadDWordLE</td>
</tr>
<tr>
<td>37.9.5</td>
<td>TStreamHelper.ReadQWordLE</td>
</tr>
<tr>
<td>37.9.6</td>
<td>TStreamHelper.WriteWordLE</td>
</tr>
<tr>
<td>37.9.7</td>
<td>TStreamHelper.WriteDWordLE</td>
</tr>
</tbody>
</table>
37.9.8 TStreamHelper.WriteQWordLE ............................................. 880
37.9.9 TStreamHelper.ReadWordBE ................................................. 881
37.9.10 TStreamHelper.ReadDWordBE .............................................. 881
37.9.11 TStreamHelper.ReadQWordBE .............................................. 881
37.9.12 TStreamHelper.WriteWordBE .............................................. 881
37.9.13 TStreamHelper.WriteQWordBE ............................................ 882
37.9.14 TStreamHelper.WriteDWordBE ............................................ 882
37.9.15 TStreamHelper.ReadSingle ................................................. 882
37.9.16 TStreamHelper.ReadDouble ............................................... 882
37.9.17 TStreamHelper.WriteSingle .............................................. 883
37.9.18 TStreamHelper.WriteDouble .............................................. 883
37.10 TStreamReader ................................................................. 883
37.10.1 Description ................................................................. 883
37.10.2 Method overview ............................................................ 883
37.10.3 Property overview .......................................................... 883
37.10.4 TStreamReader.Create ..................................................... 884
37.10.5 TStreamReader.Destroy .................................................... 884
37.10.6 TStreamReader.Reset ....................................................... 884
37.10.7 TStreamReader.Close ...................................................... 884
37.10.8 TStreamReader.ReadLine .................................................. 885
37.10.9 TStreamReader.BaseStream .............................................. 885
37.10.10 TStreamReader.OwnsStream ............................................ 885
37.11 TStringReader ................................................................. 885
37.11.1 Description ................................................................. 885
37.11.2 Method overview ............................................................ 886
37.11.3 TStringReader.Create ..................................................... 886
37.11.4 TStringReader.Destroy .................................................... 886
37.11.5 TStringReader.Reset ....................................................... 886
37.11.6 TStringReader.Close ...................................................... 887
37.11.7 TStringReader.ReadLine .................................................. 887
37.12 TTextReader ................................................................. 887
37.12.1 Description ................................................................. 887
37.12.2 Method overview ............................................................ 887
37.12.3 Property overview .......................................................... 887
37.12.4 TTextReader.Create ..................................................... 888
37.12.5 TTextReader.Reset ....................................................... 888
37.12.6 TTextReader.Close ...................................................... 888
37.12.7 TTextReader.ReadLine .................................................. 888
37.12.8 TTextReader.Eof ............................................................ 889
37.13 TWindowedStream ........................................................... 889
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>37.13.1 Description</td>
<td>889</td>
</tr>
<tr>
<td>37.13.2 Method overview</td>
<td>889</td>
</tr>
<tr>
<td>37.13.3 TWindowedStream.Create</td>
<td>889</td>
</tr>
<tr>
<td>37.13.4 TWindowedStream.Destroy</td>
<td>890</td>
</tr>
<tr>
<td>37.13.5 TWindowedStream.Read</td>
<td>890</td>
</tr>
<tr>
<td>37.13.6 TWindowedStream.Write</td>
<td>890</td>
</tr>
<tr>
<td>37.13.7 TWindowedStream.Seek</td>
<td>890</td>
</tr>
<tr>
<td><strong>38 Reference for unit 'StreamIO’</strong></td>
<td>892</td>
</tr>
<tr>
<td>38.1 Used units</td>
<td>892</td>
</tr>
<tr>
<td>38.2 Overview</td>
<td>892</td>
</tr>
<tr>
<td>38.3 Procedures and functions</td>
<td>892</td>
</tr>
<tr>
<td>38.3.1 AssignStream</td>
<td>892</td>
</tr>
<tr>
<td>38.3.2 GetStream</td>
<td>893</td>
</tr>
<tr>
<td><strong>39 Reference for unit 'syncobjs’</strong></td>
<td>894</td>
</tr>
<tr>
<td>39.1 Used units</td>
<td>894</td>
</tr>
<tr>
<td>39.2 Overview</td>
<td>894</td>
</tr>
<tr>
<td>39.3 Constants, types and variables</td>
<td>894</td>
</tr>
<tr>
<td>39.3.1 Constants</td>
<td>894</td>
</tr>
<tr>
<td>39.3.2 Types</td>
<td>894</td>
</tr>
<tr>
<td>39.4 ELockException</td>
<td>895</td>
</tr>
<tr>
<td>39.4.1 Description</td>
<td>895</td>
</tr>
<tr>
<td>39.5 ELockRecursionException</td>
<td>895</td>
</tr>
<tr>
<td>39.5.1 Description</td>
<td>895</td>
</tr>
<tr>
<td>39.6 ESyncObjectException</td>
<td>895</td>
</tr>
<tr>
<td>39.6.1 Description</td>
<td>895</td>
</tr>
<tr>
<td>39.7 TCriticalSection</td>
<td>895</td>
</tr>
<tr>
<td>39.7.1 Description</td>
<td>895</td>
</tr>
<tr>
<td>39.7.2 Method overview</td>
<td>896</td>
</tr>
<tr>
<td>39.7.3 TCriticalSection.Acquire</td>
<td>896</td>
</tr>
<tr>
<td>39.7.4 TCriticalSection.Release</td>
<td>896</td>
</tr>
<tr>
<td>39.7.5 TCriticalSection.Enter</td>
<td>897</td>
</tr>
<tr>
<td>39.7.6 TCriticalSection.TryEnter</td>
<td>897</td>
</tr>
<tr>
<td>39.7.7 TCriticalSection.Leave</td>
<td>897</td>
</tr>
<tr>
<td>39.7.8 TCriticalSection.Create</td>
<td>897</td>
</tr>
<tr>
<td>39.7.9 TCriticalSection.Destroy</td>
<td>898</td>
</tr>
<tr>
<td>39.8 TEventObject</td>
<td>898</td>
</tr>
<tr>
<td>39.8.1 Description</td>
<td>898</td>
</tr>
<tr>
<td>39.8.2 Method overview</td>
<td>898</td>
</tr>
<tr>
<td>39.8.3 Property overview</td>
<td>898</td>
</tr>
</tbody>
</table>
41.3.1 Constants .................................................. 906
41.3.2 Types ...................................................... 910
41.4 Central_File_Header_Type ........................................ 911
41.5 CodeRec ..................................................... 912
41.6 End_of_Central_Dir_Type ....................................... 912
41.7 Extensible_Data_Field_Header_Type ......................... 912
41.8 Local_File_Header_Type ........................................ 913
41.9 Zip64_End_of_Central_Dir_Locator_type .................. 913
41.10 Zip64_End_of_Central_Dir_type .......................... 913
41.11 Zip64_Extended_Info_Field_Type ......................... 914
41.12 EZipError .................................................. 914
   41.12.1 Description ........................................... 914
41.13 TCompressor ................................................. 914
   41.13.1 Description ........................................... 914
   41.13.2 Method overview ....................................... 914
   41.13.3 Property overview ..................................... 914
   41.13.4 TCompressor.Create ..................................... 915
   41.13.5 TCompressor.Compress .................................... 915
   41.13.6 TCompressor.ZipID ..................................... 915
   41.13.7 TCompressor.ZipVersionReqd ......................... 915
   41.13.8 TCompressor.ZipBitFlag ................................ 915
   41.13.9 TCompressor.Terminate ................................ 915
   41.13.10 TCompressor.BufferSize ................................ 916
   41.13.11 TCompressor.OnPercent ................................ 916
   41.13.12 TCompressor.OnProgress ................................ 916
   41.13.13 TCompressor.Crc32Val ................................ 916
   41.13.14 TCompressor.Terminated ............................. 916
41.14 TDeCompressor ................................................. 917
   41.14.1 Description ........................................... 917
   41.14.2 Method overview ....................................... 917
   41.14.3 Property overview ..................................... 917
   41.14.4 TDeCompressor.Create ..................................... 917
   41.14.5 TDeCompressor.DeCompress ............................ 917
   41.14.6 TDeCompressor.Terminate ................................ 917
   41.14.7 TDeCompressor.ZipID ..................................... 918
   41.14.8 TDeCompressor.BufferSize ................................ 918
   41.14.9 TDeCompressor.OnPercent ................................ 918
   41.14.10 TDeCompressor.OnProgress ............................ 918
   41.14.11 TDeCompressor.OnProgressEx ......................... 918
   41.14.12 TDeCompressor.Crc32Val .............................. 918
41.14.1 TDeCompressor.Terminated ........................................ 919
41.15 TDeflater ................................................................. 919
  41.15.1 Description .......................................................... 919
  41.15.2 Method overview .................................................... 919
  41.15.3 Property overview .................................................. 919
  41.15.4 TDeflater.Create ................................................... 919
  41.15.5 TDeflater.Compress ................................................ 919
  41.15.6 TDeflater.ZipID ..................................................... 920
  41.15.7 TDeflater.ZipVersionReqd ......................................... 920
  41.15.8 TDeflater.ZipBitFlag .............................................. 920
  41.15.9 TDeflater.CompressionLevel ...................................... 920
41.16 TFullZipFileEntries .................................................. 921
  41.16.1 Description .......................................................... 921
  41.16.2 Property overview .................................................. 921
  41.16.3 TFullZipFileEntries.FullEntries .................................. 921
41.17 TFullZipFileEntry ...................................................... 921
  41.17.1 Description .......................................................... 921
  41.17.2 Property overview .................................................. 921
  41.17.3 TFullZipFileEntry.BitFlags ....................................... 922
  41.17.4 TFullZipFileEntry.CompressMethod ................................ 922
  41.17.5 TFullZipFileEntry.CompressedSize ................................ 922
  41.17.6 TFullZipFileEntry.CRC32 ......................................... 922
41.18 TInflater ................................................................. 922
  41.18.1 Description .......................................................... 922
  41.18.2 Method overview .................................................... 923
  41.18.3 TInflater.Create ................................................... 923
  41.18.4 TInflater.DeCompress .............................................. 923
  41.18.5 TInflater.ZipID ..................................................... 923
41.19 TShrinker ................................................................. 923
  41.19.1 Description .......................................................... 923
  41.19.2 Method overview .................................................... 923
  41.19.3 TShrinker.Create ................................................... 924
  41.19.4 TShrinker.Destroy .................................................. 924
  41.19.5 TShrinker.Compress ................................................ 924
  41.19.6 TShrinker.ZipID ..................................................... 924
  41.19.7 TShrinker.ZipVersionReqd ......................................... 924
  41.19.8 TShrinker.ZipBitFlag .............................................. 924
41.20 TUnZipper ............................................................... 925
  41.20.1 Description .......................................................... 925
  41.20.2 Method overview .................................................... 925
41.22.5 TZipFileEntry.IsDirectory ........................................ 934
41.22.6 TZipFileEntry.IsLink ............................................. 935
41.22.7 TZipFileEntry.Assign ............................................ 935
41.22.8 TZipFileEntry.Stream ............................................ 935
41.22.9 TZipFileEntry.ArchiveFileName ................................. 935
41.22.10 TZipFileEntry.UTF8ArchiveFileName ............................ 935
41.22.11 TZipFileEntry.DiskFileName .................................... 936
41.22.12 TZipFileEntry.UTF8DiskFileName ................................ 936
41.22.13 TZipFileEntry.Size ............................................. 936
41.22.14 TZipFileEntry.DateTime ....................................... 936
41.22.15 TZipFileEntry.OS ............................................... 936
41.22.16 TZipFileEntry.Attributes ...................................... 937
41.22.17 TZipFileEntry.CompressionLevel .............................. 937

41.23 TZipper ........................................................... 937
41.23.1 Description ....................................................... 937
41.23.2 Method overview ............................................... 937
41.23.3 Property overview .............................................. 938
41.23.4 TZipper.Create ................................................. 938
41.23.5 TZipper.Destroy ............................................... 938
41.23.6 TZipper.ZipAllFiles ......................................... 938
41.23.7 TZipper.SaveToFile ............................................ 939
41.23.8 TZipper.SaveToStream ..................................... 939
41.23.9 TZipper.ZipFile ............................................... 939
41.23.10 TZipper.ZipFiles .............................................. 939
41.23.11 TZipper.Zip .................................................. 940
41.23.12 TZipper.Clear ............................................... 940
41.23.13 TZipper.Terminate ............................................ 941
41.23.14 TZipper.BufferSize ......................................... 941
41.23.15 TZipper.OnPercent ........................................... 941
41.23.16 TZipper.OnProgress .......................................... 941
41.23.17 TZipper.OnStartFile ...................................... 941
41.23.18 TZipper.OnEndFile ....................................... 941
41.23.19 TZipper.FileName ........................................... 942
41.23.20 TZipper.FileComment .................................... 942
41.23.21 TZipper.Files ............................................... 942
41.23.22 TZipper.InMemSize ...................................... 942
41.23.23 TZipper.Entries ........................................... 943
41.23.24 TZipper.Terminated ..................................... 943
41.23.25 TZipper.UseLanguageEncoding ............................ 943
42 Reference for unit 'ZStream' 944
42.1 Used units ........................................................................ 944
42.2 Overview ........................................................................ 944
42.3 Constants, types and variables ............................................ 944
  42.3.1 Types ......................................................................... 944
42.4 Ecompressionerror .............................................................. 945
  42.4.1 Description .................................................................. 945
42.5 Edecompressionerror ............................................................ 945
  42.5.1 Description .................................................................. 945
42.6 Egzfileerror ........................................................................ 945
  42.6.1 Description .................................................................. 945
42.7 Ezliberror ........................................................................... 945
  42.7.1 Description .................................................................. 945
42.8 Tcompressionstream ........................................................... 945
  42.8.1 Description .................................................................. 945
  42.8.2 Method overview ......................................................... 946
  42.8.3 Property overview ....................................................... 946
  42.8.4 Tcompressionstream.create ......................................... 946
  42.8.5 Tcompressionstream.destroy ....................................... 946
  42.8.6 Tcompressionstream.write ........................................ 946
  42.8.7 Tcompressionstream.flush ......................................... 947
  42.8.8 Tcompressionstream.get_compressionrate ................. 947
  42.8.9 Tcompressionstream.OnProgress ............................... 947
42.9 Tcustomzlibstream .............................................................. 947
  42.9.1 Description .................................................................. 947
  42.9.2 Method overview ....................................................... 948
  42.9.3 Tcustomzlibstream.create ......................................... 948
  42.9.4 Tcustomzlibstream.destroy ....................................... 948
42.10 Tdecompressionstream ...................................................... 948
  42.10.1 Description ............................................................... 948
  42.10.2 Method overview ................................................... 948
  42.10.3 Property overview .................................................. 949
  42.10.4 Tdecompressionstream.create ................................. 949
  42.10.5 Tdecompressionstream.destroy ............................... 949
  42.10.6 Tdecompressionstream.read ..................................... 949
  42.10.7 Tdecompressionstream.Seek ..................................... 950
  42.10.8 Tdecompressionstream.get_compressionrate ........ 950
  42.10.9 Tdecompressionstream.OnProgress ....................... 950
42.11 TGZFileStream ................................................................. 951
  42.11.1 Description ............................................................. 951
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>42.11.2</td>
<td>Method overview</td>
<td>951</td>
</tr>
<tr>
<td>42.11.3</td>
<td>TGZFileStream.create</td>
<td>951</td>
</tr>
<tr>
<td>42.11.4</td>
<td>TGZFileStream.read</td>
<td>951</td>
</tr>
<tr>
<td>42.11.5</td>
<td>TGZFileStream.write</td>
<td>952</td>
</tr>
<tr>
<td>42.11.6</td>
<td>TGZFileStream.seek</td>
<td>952</td>
</tr>
<tr>
<td>42.11.7</td>
<td>TGZFileStream.destroy</td>
<td>952</td>
</tr>
</tbody>
</table>
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 ’ascii85’

1.1 Used units

Table 1.1: Used units by unit ’ascii85’

<table>
<thead>
<tr>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Classes</td>
<td>??</td>
</tr>
<tr>
<td>System</td>
<td>??</td>
</tr>
<tr>
<td>sysutils</td>
<td>??</td>
</tr>
</tbody>
</table>

1.2 Overview

The ascii85 provides an ASCII 85 or base 85 decoding algorithm. It is class and stream based: the TASCII85DecoderStream (90) 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 (624) or base64 (118) units.

1.3 Constants, types and variables

1.3.1 Types

TASCII85State = (ascInitial,ascOneEncodedChar,ascTwoEncodedChars,
                 ascThreeEncodedChars,ascFourEncodedChars,
                 ascNoEncodedChar,ascPrefix)
Table 1.2: Enumeration values for type TASCII85State

<table>
<thead>
<tr>
<th>Value</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>ascFourEncodedChars</td>
<td>Four encoded characters in buffer.</td>
</tr>
<tr>
<td>ascInitial</td>
<td>Initial state</td>
</tr>
<tr>
<td>ascNoEncodedChar</td>
<td>No encoded characters in buffer.</td>
</tr>
<tr>
<td>ascOneEncodedChar</td>
<td>One encoded character in buffer.</td>
</tr>
<tr>
<td>ascPrefix</td>
<td>Prefix processing</td>
</tr>
<tr>
<td>ascThreeEncodedChars</td>
<td>Three encoded characters in buffer.</td>
</tr>
<tr>
<td>ascTwoEncodedChars</td>
<td>Two encoded characters in buffer.</td>
</tr>
</tbody>
</table>

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

1.4 TASCII85DecoderStream

1.4.1 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.

1.4.2 Method overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>91</td>
<td>Close</td>
<td>Close decoder</td>
</tr>
<tr>
<td>91</td>
<td>ClosedP</td>
<td>Check if the state is correct</td>
</tr>
<tr>
<td>90</td>
<td>Create</td>
<td>Create new ASCII 85 decoder stream</td>
</tr>
<tr>
<td>91</td>
<td>Decode</td>
<td>Decode source byte</td>
</tr>
<tr>
<td>91</td>
<td>Destroy</td>
<td>Clean up instance</td>
</tr>
<tr>
<td>92</td>
<td>Read</td>
<td>Read data from stream</td>
</tr>
<tr>
<td>92</td>
<td>Seek</td>
<td>Set stream position</td>
</tr>
</tbody>
</table>

1.4.3 Property overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Properties</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>92</td>
<td>BExpectBoundary</td>
<td>rw</td>
<td>Expect character</td>
</tr>
</tbody>
</table>

1.4.4 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 (91)
1.4.5 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 (91)

1.4.6 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 (91), TASCII85DecoderStream.Read (92), TASCII85DecoderStream.Destroy (91)

1.4.7 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 (91), TASCII85DecoderStream.BExpectBoundary (92)

1.4.8 TASCII85DecoderStream.Destroy

Synopsis: Clean up instance

Declaration: destructor Destroy; Override

Visibility: public

Description: Destroy closes the input stream using Close (91) and cleans up the TASCII85DecoderStream instance from memory.

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

See also: TASCII85DecoderStream.Close (91)
1.4.9 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.

1.4.10 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 (92)

1.4.11 TASCII85DecoderStream.BExpectBoundary

Synopsis: Expect character

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 (91)

1.5 TASCII85EncoderStream

1.5.1 Description

TASCII85EncoderStream is the counterpart to the TASCII85DecoderStream (90) decoder stream: what TASCII85EncoderStream encodes, can be decoded by TASCII85DecoderStream (90). 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 (93), TASCII85DecoderStream (90)
1.5.2 Method overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>93</td>
<td>Create</td>
<td>Create a new instance of TASCII85EncoderStream</td>
</tr>
<tr>
<td>93</td>
<td>Destroy</td>
<td>Flushed the data to the output stream and cleans up the encoder instance.</td>
</tr>
<tr>
<td>93</td>
<td>Write</td>
<td>Write data encoded to the destination stream</td>
</tr>
</tbody>
</table>

1.5.3 Property overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Properties</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>94</td>
<td>Boundary</td>
<td>r</td>
<td>Is a boundary delineator written before and after the data</td>
</tr>
<tr>
<td>94</td>
<td>Width</td>
<td>r</td>
<td>Width of the lines written to the data stream</td>
</tr>
</tbody>
</table>

1.5.4 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 (92), Width (94), Boundary (94)

1.5.5 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 (93), TASCII85EncoderStream.Boundary (94)

1.5.6 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 (93)
1.5.7  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 (94), Create (93)

1.5.8  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 (94), Create (93)

1.6  TASCII85RingBuffer

1.6.1  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 (90) decoder stream will create an instance of this class automatically.

See also: TASCII85DecoderStream (90)

1.6.2  Method overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>95</td>
<td>Read</td>
<td>Read data from the internal buffer</td>
</tr>
<tr>
<td>95</td>
<td>Write</td>
<td>Write data to the internal buffer</td>
</tr>
</tbody>
</table>

1.6.3  Property overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Properties</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>95</td>
<td>FillCount</td>
<td>r</td>
<td>Number of bytes in buffer</td>
</tr>
<tr>
<td>95</td>
<td>Size</td>
<td>r</td>
<td>Size of buffer</td>
</tr>
</tbody>
</table>
1.6.4 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 (95), TASCII85RingBuffer.Read (95), TASCII85RingBuffer.Size (95)

1.6.5 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 (95), TASCII85RingBuffer.Write (95), TASCII85RingBuffer.Size (95)

1.6.6 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 (95), TASCII85RingBuffer.Read (95), TASCII85RingBuffer.Size (95)

1.6.7 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 (95)
Chapter 2

Reference for unit ’AVL_Tree’

2.1 Used units

<table>
<thead>
<tr>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Classes</td>
<td>??</td>
</tr>
<tr>
<td>System</td>
<td>??</td>
</tr>
<tr>
<td>sysutils</td>
<td>??</td>
</tr>
</tbody>
</table>

2.2 Overview

The avl_tree unit implements a general-purpose AVL (balanced) tree class: the TAVLTree (97) class and its associated data node class TAVLTreeNode (111).

2.3 Constants, types and variables

2.3.1 Types

PAVLTreeNode = ^TAVLTreeNode

Pointer to TAVLTreeNode

TAVLTreeClass = Class of TAVLTree

TAVLTreeClass is the class of TAVLTree (97).

TAVLTreeNodeClass = Class of TAVLTreeNode

TAVLTreeNodeClass is the class of TAVLTreeNode (111). It is the type of the TAVLTree.NodeClass (110) property and determines what class of nodes will be created by the tree.

T TObjectSortCompare = function(Tree: TAVLTree; Data1: Pointer; Data2: Pointer) : Integer of object
TObjectSortCompare is the prototype for the TAVLTree.OnObjectCompare (110) 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 (110): it also passes the tree to the sort mechanism.

### 2.3.2 Variables

**NodeMemManager** : TAVLTreeNodeMemManager

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

### 2.4 TAVLTree

#### 2.4.1 Description

TAVLTree maintains a balanced AVL tree. The tree consists of TAVLTreeNode (111) 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 (110) property.

See also: TAVLTreeNode (111)
## Method overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>Add</td>
<td>Add a new node to the tree</td>
</tr>
<tr>
<td>101</td>
<td>AddAscendingSequence</td>
<td>Assign another tree</td>
</tr>
<tr>
<td>104</td>
<td>Assign</td>
<td>Clear the tree</td>
</tr>
<tr>
<td>104</td>
<td>Compare</td>
<td>Compare 2 nodes</td>
</tr>
<tr>
<td>109</td>
<td>ConsistencyCheck</td>
<td>Check the consistency of the tree</td>
</tr>
<tr>
<td>99</td>
<td>Create</td>
<td>Create a new instance of TAVLTree</td>
</tr>
<tr>
<td>99</td>
<td>CreateObjectCompare</td>
<td>Create an instance of the tree with extended compare method</td>
</tr>
<tr>
<td>101</td>
<td>Delete</td>
<td>Delete a node from the tree</td>
</tr>
<tr>
<td>99</td>
<td>Destroy</td>
<td>Destroy the TAVLTree instance</td>
</tr>
<tr>
<td>100</td>
<td>DisposeNode</td>
<td>Dispose of a node outside of the tree</td>
</tr>
<tr>
<td>103</td>
<td>Equals</td>
<td>Check if two trees are equal</td>
</tr>
<tr>
<td>104</td>
<td>Find</td>
<td>Find a data item in the tree</td>
</tr>
<tr>
<td>106</td>
<td>FindHighest</td>
<td>Find the highest (rightmost) node in the tree</td>
</tr>
<tr>
<td>104</td>
<td>FindKey</td>
<td>Find a data item in the tree using alternate compare mechanism</td>
</tr>
<tr>
<td>107</td>
<td>FindLeftMost</td>
<td>Find the node most left to a specified data node</td>
</tr>
<tr>
<td>107</td>
<td>FindLeftMostKey</td>
<td>Find the node most left to a specified key node</td>
</tr>
<tr>
<td>108</td>
<td>FindLeftMostSameKey</td>
<td>Find the node most left to a specified node with the same data</td>
</tr>
<tr>
<td>106</td>
<td>FindLowest</td>
<td>Find the lowest (leftmost) node in the tree</td>
</tr>
<tr>
<td>106</td>
<td>FindNearest</td>
<td>Find the node closest to the data in the tree</td>
</tr>
<tr>
<td>105</td>
<td>FindNearestKey</td>
<td>Find nearest key for a data pointer</td>
</tr>
<tr>
<td>106</td>
<td>FindPointer</td>
<td>Search for a data pointer</td>
</tr>
<tr>
<td>105</td>
<td>FindPrecessor</td>
<td></td>
</tr>
<tr>
<td>107</td>
<td>FindRightMost</td>
<td>Find the node most right to a specified node</td>
</tr>
<tr>
<td>107</td>
<td>FindRightMostKey</td>
<td>Find the node most right to a specified key node</td>
</tr>
<tr>
<td>108</td>
<td>FindRightMostSameKey</td>
<td>Find the node most right of a specified node with the same data</td>
</tr>
<tr>
<td>105</td>
<td>FindSuccessor</td>
<td>Find successor to node</td>
</tr>
<tr>
<td>103</td>
<td>FreeAndClear</td>
<td>Clears the tree and frees nodes</td>
</tr>
<tr>
<td>103</td>
<td>FreeAndDelete</td>
<td>Delete a node from the tree and destroy it</td>
</tr>
<tr>
<td>108</td>
<td>GetEnumerator</td>
<td>Get an enumerator for the tree</td>
</tr>
<tr>
<td>108</td>
<td>GetEnumeratorHighToLow</td>
<td>Return an enumerator that enumerates the tree in reversed order</td>
</tr>
<tr>
<td>103</td>
<td>IsEqual</td>
<td>Check whether 2 tree instances are equal</td>
</tr>
<tr>
<td>102</td>
<td>MoveDataLeftMost</td>
<td>Move data to the nearest left element</td>
</tr>
<tr>
<td>102</td>
<td>MoveDataRightMost</td>
<td>Move data to the nearest right element</td>
</tr>
<tr>
<td>100</td>
<td>NewNode</td>
<td>Create a new tree node</td>
</tr>
<tr>
<td>109</td>
<td>NodeToReportStr</td>
<td>Create a textual dump of the tree</td>
</tr>
<tr>
<td>101</td>
<td>Remove</td>
<td>Remove a data item from the list</td>
</tr>
<tr>
<td>102</td>
<td>RemovePointer</td>
<td>Remove a pointer item from the list</td>
</tr>
<tr>
<td>109</td>
<td>ReportAsString</td>
<td>Return the tree report as a string</td>
</tr>
<tr>
<td>99</td>
<td>SetNodeManager</td>
<td>Set the node instance manager to use</td>
</tr>
<tr>
<td>109</td>
<td>WriteReportToStream</td>
<td>Write the contents of the tree consistency check to the stream</td>
</tr>
</tbody>
</table>
2.4.3 Property overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Properties</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>111</td>
<td>Count</td>
<td>r</td>
<td>Number of nodes in the tree.</td>
</tr>
<tr>
<td>110</td>
<td>NodeClass</td>
<td>rw</td>
<td>Node class to create</td>
</tr>
<tr>
<td>110</td>
<td>OnCompare</td>
<td>rw</td>
<td>Compare function used when comparing nodes</td>
</tr>
<tr>
<td>110</td>
<td>OnManagedObject</td>
<td>rw</td>
<td>Compare handler</td>
</tr>
<tr>
<td>110</td>
<td>Root</td>
<td>r</td>
<td>Root node of the tree</td>
</tr>
</tbody>
</table>

2.4.4 TAVLTree.Create

Synopsis: Create a new instance of TAVLTree

Declaration:

```delphi
class TAVLTree
constructor Create(const OnCompareMethod: TListSortCompare);
constructor Create
visibility: public

Description: Create initializes a new instance of TAVLTree (97). An alternate OnCompare (110) can be provided: the default OnCompare method compares the 2 data pointers of a node.

See also: OnCompare (110)
```

2.4.5 TAVLTree.CreateObjectCompare

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

Declaration:

```delphi
class TAVLTree
constructor CreateObjectCompare(const OnCompareMethod: TObjectSortCompare);
visibility: public

Description: CreateObjectCompare is an alternative constructor that accepts a TObjectSortCompare (97) compare function instead of a regular TListSortCompare compare function. The compare function can still be set in the TAVLTree.OnObjectCompare (110) property.

See also: TAVLTree.OnObjectCompare (110)
```

2.4.6 TAVLTree.Destroy

Synopsis: Destroy the TAVLTree instance

Declaration:

```delphi
class TAVLTree
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 (102), TAVLTree.Create (99)
```

2.4.7 TAVLTree.SetNodeManager

Synopsis: Set the node instance manager to use

Declaration:

```delphi
class TAVLTree
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 manage the nodes of multiple `TAVLTree` instances.

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

See also: `TBaseAVLTreeNodeManager`, `TAVLTreeNodeMemManager`  

### 2.4.8 `TAVLTree.NewNode`

**Synopsis:** Create a new tree node

**Declaration:**

```plaintext
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` class is used to create a new node.

See also: `TAVLTree.NodeClass`, `TAVLTree.Add`, `TAVLTree.DisposeNode`

### 2.4.9 `TAVLTree.DisposeNode`

**Synopsis:** Dispose of a node outside of the tree

**Declaration:**

```plaintext
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`

### 2.4.10 `TAVLTree.Add`

**Synopsis:** Add a new node to the tree

**Declaration:**

```plaintext
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`, `TAVLTree.Remove`
2.4.11 TAVLTree.AddAscendingSequence

Synopsis:

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

Visibility: public

Description: AddAscendingSequence is an optimized version of Add (100) 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 (100)

2.4.12 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 (114) 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 (101), TAVLTree.RemovePointer (102), TAVLTree.Clear (102)

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

See also: TAVLTree.Delete (101), TAVLTree.RemovePointer (102), TAVLTree.Clear (102), TAVLTree.Find (104)
2.4.14 TAVLTree.RemovePointer

Synopsis: Remove a pointer item from the list.

Declaration: function RemovePointer(Data: Pointer) : Boolean

Visibility: public

Description: Remove uses FindPointer (106) 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 (101), TAVLTree.Delete (101), TAVLTree.Clear (102)

2.4.15 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 (102)

2.4.16 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 (102)

2.4.17 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 (103) instead.

See also: TAVLTree.FreeAndClear (103), TAVLTree.Delete (101)
2.4.18 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 (102), TAVLTree.Delete (101), TAVLTree.FreeAndDelete (103)

2.4.19 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 (102), TAVLTree.Delete (101), TAVLTree.FreeAndClear (103)

2.4.20 TAVLTree.Equals
Synopsis: Check if two trees are equal
Declaration: function Equals(Obj: TObject) : Boolean; Override
   Visibility: public
Description: Equals checks, when passed an TA VLTtree, whether the tree is equal (using TAvlTree.IsEqual (103), comparing keys only). If another type of object is passed, the inherited IsEqual is called.
   Errors: None.
See also: TAvlTree.IsEqual (103)

2.4.21 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 (103)
2.4.22 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 (103)

2.4.23 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 (110) or TAVLTree.OnObjectCompare (110) 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 (110), TAVLTree.OnCompare (110)

2.4.24 TAVLTree.Find

Synopsis: Find a data item in the tree.

Declaration: function Find(Data: Pointer) : TAVLTreeNode

Visibility: public

Description: Find uses the default OnCompare (110) 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 (106).

See also: OnCompare (110), FindKey (104)

2.4.25 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
CHAPTER 2. REFERENCE FOR UNIT 'AVL_TREE'

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 (110), Find (104)

2.4.26 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 (104)

2.4.27 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 (105), TAVLTree.MoveDataRightMost (102)

2.4.28 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 (105), TAVLTree.MoveDataLeftMost (102)
2.4.29 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 (106)

2.4.30 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 (106)

2.4.31 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 (106), FindLowest (106), Find (104), FindKey (104)

2.4.32 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 (104), where a custom compare function can be used.
The default OnCompare (110) compares the data pointers, so the default Find will return the same node as FindPointer
See also: TAVLTree.Find (104), TAVLTree.FindKey (104)
2.4.33 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 (107), TAVLTree.FindLeftMostKey (107), TAVLTree.FindRightMostKey (107)

2.4.34 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 (107), TAVLTree.FindLeftMostKey (107), TAVLTree.FindRightMostKey (107)

2.4.35 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 (107), TAVLTree.FindRightMost (107), TAVLTree.FindRightMostKey (107)

2.4.36 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 (107), TAVLTree.FindRightMost (107), TAVLTree.FindLeftMostKey (107)
2.4.37 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: FindLeftMostSameKey finds the node most left from and with the same data as the specified node ANode.
See also: TAVLTree.FindLeftMost (107), TAVLTree.FindLeftMostKey (107), TAVLTree.FindRightMostSameKey (108)

2.4.38 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: FindRightMostSameKey finds the node most right from and with the same data as the specified node ANode.
See also: TAVLTree.FindRightMost (107), TAVLTree.FindRightMostKey (107), TAVLTree.FindLeftMostSameKey (108)

2.4.39 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 (112).
See also: TAVLTreeNodeEnumerator (112)

2.4.40 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 (108)
2.4.41 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 (109)

2.4.42 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 (109)

2.4.43 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 (109) 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 (109)

2.4.44 TAVLTree.ReportAsString

Synopsis: Return the tree report as a string

Declaration: function ReportAsString : string

Visibility: public

Description: ReportAsString calls WriteReportToStream (109) and returns the stream data as a string.

See also: TAVLTree.WriteReportToStream (109)
2.4.45 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 (99)

2.4.46 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 (110) is not set.
See also: TAVLTree.OnCompare (110)

2.4.47 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 (100) will use this class when creating a new node. This can be set to a descendent class of TAVLTreeNode (111), but not if there are already nodes in the tree.
See also: TAVLTreeNode (111), TAVLTree.NewNode (100)

2.4.48 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 (100), Delete (101), Remove (101), RemovePointer (102), or Clear (102) methods to manipulate the items in the tree.
See also: TAVLTree.Add (100), TAVLTree.Delete (101), TAVLTree.Remove (101), TAVLTree.RemovePointer (102), TAVLTree.Clear (102)
2.4.49  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.

2.5  TAVLTreeNode

2.5.1  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 (97), TAVLTreeNode.Data (??)

2.5.2  Method overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>112</td>
<td>Clear</td>
<td>Clears the node’s data</td>
</tr>
<tr>
<td>112</td>
<td>ConsistencyCheck</td>
<td>Check consistency of the node and below nodes.</td>
</tr>
<tr>
<td>112</td>
<td>GetCount</td>
<td>Get the number of nodes</td>
</tr>
<tr>
<td>111</td>
<td>Precessor</td>
<td>Preceding node in the tree</td>
</tr>
<tr>
<td>111</td>
<td>Successor</td>
<td>Succeeding node in the tree</td>
</tr>
<tr>
<td>112</td>
<td>TreeDepth</td>
<td>Level of the node in the tree below</td>
</tr>
</tbody>
</table>

2.5.3  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.Precursor (111)

2.5.4  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 (111)
2.5.5 **TAVLTreeNode.Clear**

**Synopsis:** Clears the node’s data

**Declaration:**

```delphi
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.

2.5.6 **TAVLTreeNode.TreeDepth**

**Synopsis:** Level of the node in the tree below

**Declaration:**

```delphi
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 (??)

2.5.7 **TAVLTreeNode.ConsistencyCheck**

**Synopsis:** Check consistency of the node and below nodes.

**Declaration:**

```delphi
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.

2.5.8 **TAVLTreeNode.GetCount**

**Synopsis:** Get the number of nodes

**Declaration:**

```delphi
function GetCount : SizeInt;
```

**Visibility:** public

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

**Errors:** None.

2.6 **TAVLTreeNodeEnumerator**

2.6.1 **Description**

`TAVLTreeNodeEnumerator` is a class which implements the enumerator interface for the TAVL-Tree (97). It enumerates all the nodes in the tree.

**See also:** TAVLTree (97)
2.6.2 Method overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>113</td>
<td>Create</td>
<td>Create a new instance of TAVLTreeNodeEnumerator</td>
</tr>
<tr>
<td>113</td>
<td>GetEnumerator</td>
<td>Returns the enumerator</td>
</tr>
<tr>
<td>113</td>
<td>MoveNext</td>
<td>Move to next node in the tree</td>
</tr>
</tbody>
</table>

2.6.3 Property overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Properties</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>113</td>
<td>Current</td>
<td>r</td>
<td>Current node in the tree</td>
</tr>
<tr>
<td>114</td>
<td>LowToHigh</td>
<td>r</td>
<td>Should the enumerator return nodes from low to high or high to low</td>
</tr>
</tbody>
</table>

2.6.4 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.

2.6.5 TAVLTreeNodeEnumerator.GetEnumerator

Synopsis: Returns the enumerator

Declaration: function GetEnumerator : TAVLTreeNodeEnumerator

Visibility: public

Description: GetEnumerator returns Self..

2.6.6 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 (105).

See also: TAVLTree.FindSuccessor (105)

2.6.7 TAVLTreeNodeEnumerator.Current

Synopsis: Current node in the tree

Declaration: Property Current : TAVLTreeNode

Visibility: public

Access: Read
CHAPTER 2. REFERENCE FOR UNIT 'AVL_TREE'

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

**See also:** TAVLTreeNodeEnumerator.MoveNext (113)

---

**2.6.8 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 (113)

---

**2.7 TAVLTreeNodeMemManager**

**2.7.1 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 (111), TAVLTree (97)

---

**2.7.2 Method overview**

<table>
<thead>
<tr>
<th>Page</th>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>115</td>
<td>Clear</td>
<td>Frees all unused nodes</td>
</tr>
<tr>
<td>115</td>
<td>Create</td>
<td>Create a new instance of TAVLTreeNodeMemManager</td>
</tr>
<tr>
<td>115</td>
<td>Destroy</td>
<td></td>
</tr>
<tr>
<td>114</td>
<td>DisposeNode</td>
<td>Return a node to the free list</td>
</tr>
<tr>
<td>115</td>
<td>NewNode</td>
<td>Create a new TAVLTreeNode instance</td>
</tr>
</tbody>
</table>

---

**2.7.3 Property overview**

<table>
<thead>
<tr>
<th>Page</th>
<th>Properties</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>116</td>
<td>Count</td>
<td>r</td>
<td>Number of nodes in the list.</td>
</tr>
<tr>
<td>116</td>
<td>MaximumFreeNodeRatio</td>
<td>rw</td>
<td>Maximum amount of free nodes in the list</td>
</tr>
<tr>
<td>116</td>
<td>MinimumFreeNode</td>
<td>rw</td>
<td>Minimum amount of free nodes to be kept.</td>
</tr>
</tbody>
</table>

---

**2.7.4 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 (115)

2.7.5 TAVLTreeNodeMemManager.NewNode
Synopsis: Create a new TAVLTreeNode instance
Declaration: function NewNode : TAVLTreeNode; Override
Visibility: public
Description: NewNode returns a new TAVLTreeNode (111) 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 (114)

2.7.6 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 (116), TAVLTreeNodeMemManager.MaximumFreeNodeRatio (116)

2.7.7 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 (115)

2.7.8 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 (115)
2.7.9 **TAVLTreeNodeMemManager.MinimumFreeNode**

**Synopsis:** Minimum amount of free nodes to be kept.

**Declaration:**

```properties
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 (116)

---

2.7.10 **TAVLTreeNodeMemManager.MaximumFreeNodeRatio**

**Synopsis:** Maximum amount of free nodes in the list

**Declaration:**

```properties
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 (116), TAVLTreeNodeMemManager.MinimumFreeNode (116)

---

2.7.11 **TAVLTreeNodeMemManager.Count**

**Synopsis:** Number of nodes in the list.

**Declaration:**

```properties
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 (116), TAVLTreeNodeMemManager.MaximumFreeNodeRatio (116)

---

2.8 **TBaseAVLTreeNodeManager**

2.8.1 **Description**

TBaseAVLTreeNodeManager is an abstract class from which a descendent can be created that manages creating and disposing of tree nodes (instances of TAVLTreeNode (111)) for a TAVLTree (97) 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 (114).

The TAVLTree.SetNodeManager (99) method can be used to set the node manager that a TAVLTree instance should use.

See also: TAVLTreeNodeMemManager (114), TAVLTree.SetNodeManager (99), TAVLTreeNode (111)
### 2.8.2 Method overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>117</td>
<td>DisposeNode</td>
<td>Called when the AVL tree no longer needs node</td>
</tr>
<tr>
<td>117</td>
<td>NewNode</td>
<td>Called when the AVL tree needs a new node</td>
</tr>
</tbody>
</table>

### 2.8.3 TBaseAVLTreeNodeManager.DisposeNode

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

**Declaration:**

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

**Visibility:** public

**Description:**

DisposeNode is called by TAVLTree (97) when it no longer needs a TAVLTreeNode (111) instance. The manager may decide to re-use the instance for later use instead of destroying it.

**See also:** TBaseAVLTreeNodeManager.NewNode (117), TAVLTree.Delete (101), TAVLTreeNode (111)

### 2.8.4 TBaseAVLTreeNodeManager.NewNode

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

**Declaration:**

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

**Visibility:** public

**Description:**

NewNode is called by TAVLTree (97) when it needs a new node in TAVLTree.Add (100). It must be implemented by descendants to return a new TAVLTreeNode (111) instance.

**See also:** TBaseAVLTreeNodeManager.DisposeNode (117), TAVLTree.Add (100), TAVLTreeNode (111)
Chapter 3

Reference for unit ’base64’

3.1 Used units

<table>
<thead>
<tr>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Classes</td>
<td>??</td>
</tr>
<tr>
<td>System</td>
<td>??</td>
</tr>
<tr>
<td>sysutils</td>
<td>??</td>
</tr>
</tbody>
</table>

3.2 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 (122) for encoding, and TBase64DecodingStream (119) 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...

3.3 Constants, types and variables

3.3.1 Types

TBase64DecodingMode = (bdmStrict,bdmMIME)

<table>
<thead>
<tr>
<th>Value</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>bdmMIME</td>
<td>MIME encoding</td>
</tr>
<tr>
<td>bdmStrict</td>
<td>Strict encoding</td>
</tr>
</tbody>
</table>
**CHAPTER 3. REFERENCE FOR UNIT 'BASE64'**

`TBase64DecodingMode` determines the decoding algorithm used by `TBase64DecodingStream` (119). 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` (119) 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.

### 3.4 Procedures and functions

#### 3.4.1 DecodeStringBase64

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

**Declaration:**
```haskell
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` (119) to do this. The `Strict` parameter is passed on to the constructor as `bdmStrict` or `bdmMIME`.

See also: `DecodeStringBase64` (119), `TBase64DecodingStream` (119)

#### 3.4.2 EncodeStringBase64

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

**Declaration:**
```haskell
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` (122) to do this.

See also: `DecodeStringBase64` (119), `TBase64EncodingStream` (122)

### 3.5 EBase64DecodingException

#### 3.5.1 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.

### 3.6 TBase64DecodingStream

#### 3.6.1 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 (122)

### 3.6.2 Method overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>120</td>
<td>Create</td>
<td>Create a new instance of the TBase64DecodingStream class</td>
</tr>
<tr>
<td>120</td>
<td>Read</td>
<td>Read and decrypt data from the source stream</td>
</tr>
<tr>
<td>120</td>
<td>Reset</td>
<td>Reset the stream</td>
</tr>
<tr>
<td>121</td>
<td>Seek</td>
<td>Set stream position.</td>
</tr>
</tbody>
</table>

### 3.6.3 Property overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Properties</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>121</td>
<td>EOF</td>
<td>r</td>
<td></td>
</tr>
<tr>
<td>121</td>
<td>Mode</td>
<td>rw</td>
<td>Decoding mode</td>
</tr>
</tbody>
</table>

### 3.6.4 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 (122), TBase64DecodingMode (118)

### 3.6.5 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 (121), TBase64DecodingStream.Read (120)

### 3.6.6 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 (121), TStream.Read (??)

3.6.7 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 (120), TBase64EncodingStream.Seek (123), TStream.Seek (??)

3.6.8 TBase64DecodingStream.EOF

Synopsis:

Declaration: Property EOF : Boolean

Visibility: public
Access: Read

Description:

3.6.9 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 (119)
3.7 TBase64EncodingStream

3.7.1 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 contain all output: input must be a multiple of 3. In order to be sure that the output contains all encoded bytes, the Flush (122) 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 (119)

3.7.2 Method overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>122</td>
<td>Destroy</td>
<td>Remove a TBase64EncodingStream instance from memory</td>
</tr>
<tr>
<td>122</td>
<td>Flush</td>
<td>Flush the remaining bytes to the output stream.</td>
</tr>
<tr>
<td>123</td>
<td>Seek</td>
<td>Position the stream</td>
</tr>
<tr>
<td>123</td>
<td>Write</td>
<td>Write data to the stream.</td>
</tr>
</tbody>
</table>

3.7.3 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.

3.7.4 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 (122)
3.7.5 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 (123), TStream.Write (??)

3.7.6 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 don’t change the current file pointer position. The encryption stream is not seekable.

Errors: An EStreamError error is raised.

See also: TBase64EncodingStream.Write (123), TStream.Seek (??)
Chapter 4

Reference for unit ’BlowFish’

4.1 Used units

Table 4.1: Used units by unit ’BlowFish’

<table>
<thead>
<tr>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Classes</td>
<td>??</td>
</tr>
<tr>
<td>System</td>
<td>??</td>
</tr>
<tr>
<td>sysutils</td>
<td>??</td>
</tr>
</tbody>
</table>

4.2 Overview

The BlowFish implements a class TBlowFish (125) to handle Blowfish encryption/decryption of memory buffers, and 2 TStream (??) descendents TBlowFishDeCryptStream (126) which decrypts any data that is read from it on the fly, as well as TBlowFishEnCryptStream (127) which encrypts the data that is written to it on the fly.

4.3 Constants, types and variables

4.3.1 Constants

BFRounds = 16

Number of rounds in Blowfish encryption.

4.3.2 Types

PBlowFishKey = ^TBlowFishKey

PBlowFishKey is a simple pointer to a TBlowFishKey (125) array.

TBFBlock = Array[0..1] of LongInt
TBFBlob is the basic data structure used by the encrypting/decrypting routines in TBlowFish (125), TBlowFishDeCryptStream (126) and TBlowFishEnCryptStream (127). It is the basic encryption/decryption block for all encrypting/decrypting: all encrypting/decrypting happens on a TBFBlob structure.

TBlowFishKey = Array[0..55] of Byte

TBlowFishKey is a data structure which keeps the encryption or decryption key for the TBlowFish (125), TBlowFishDeCryptStream (126) and TBlowFishEnCryptStream (127) classes. It should be filled with the encryption key and passed to the constructor of one of these classes.

4.4 EBlowFishError

4.4.1 Description

EBlowFishError is used by the TBlowFishStream (129), TBlowFishEncryptStream (127) and TBlowFishDecryptStream (126) classes to report errors.

See also: TBlowFishStream (129), TBlowFishEncryptStream (127), TBlowFishDecryptStream (126)

4.5 TBlowFish

4.5.1 Description

TBlowFish is a simple class that can be used to encrypt/decrypt a single TBFBlob (125) data block with the Encrypt (126) and Decrypt (126) calls. It is used internally by the TBlowFishEnCryptStream (127) and TBlowFishDeCryptStream (126) classes to encrypt or decrypt the actual data.

See also: TBlowFishEnCryptStream (127), TBlowFishDeCryptStream (126)

4.5.2 Method overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>125</td>
<td>Create</td>
<td>Create a new instance of the TBlowFish class</td>
</tr>
<tr>
<td>126</td>
<td>Decrypt</td>
<td>Decrypt a block</td>
</tr>
<tr>
<td>126</td>
<td>Encrypt</td>
<td>Encrypt a block</td>
</tr>
</tbody>
</table>

4.5.3 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 (126) and Decrypt (126).

See also: Encrypt (126), Decrypt (126)
4.5.4 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 (125) specified when the TBlowFish instance was created.

See also: TBlowFishKey (125), Decrypt (126), Create (125)

4.5.5 TBlowFish.Decrypt

Synopsis: Decrypt a block

Declaration: procedure Decrypt(var Block: TBFBlock)

Visibility: public

Description: Decrypt decrypts the data in Block (always 8 bytes) using the key (125) specified when the TBlowFish instance was created. The data must have been encrypted with the same key and the Encrypt (126) call.

See also: TBlowFishKey (125), Encrypt (126), Create (125)

4.6 TBlowFishDecryptStream

4.6.1 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 (127) stream must be used.

See also: Create (129), TBlowFishEncryptStream (127)

4.6.2 Method overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>126</td>
<td>Create</td>
<td>Constructor for the class instance</td>
</tr>
<tr>
<td>127</td>
<td>Read</td>
<td>Read data from the stream</td>
</tr>
<tr>
<td>127</td>
<td>Seek</td>
<td>Set the stream position.</td>
</tr>
</tbody>
</table>

4.6.3 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 (127), TBlowFishDeCryptStream.Seek (127), TBlowFishEncryptStream.Write (128)

4.6.4 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 (129), TBlowFishEncryptStream (127)

4.6.5 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 (125) exception will be raised.

See also: Read (127), EBlowFishError (125)

4.7 TBlowFishEncryptStream

4.7.1 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 (126) stream must be used.

See also: Create (129), TBlowFishDecryptStream (126)
4.7.2 Method overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>128</td>
<td>Destroy</td>
<td>Free the TBlowFishEncryptStream</td>
</tr>
<tr>
<td>129</td>
<td>Flush</td>
<td>Flush the encryption buffer</td>
</tr>
<tr>
<td>128</td>
<td>Seek</td>
<td>Set the position in the stream</td>
</tr>
<tr>
<td>128</td>
<td>Write</td>
<td>Write data to the stream</td>
</tr>
</tbody>
</table>

4.7.3 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 (129), Create (129)

4.7.4 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 (129) mechanism can be used to write the remaining bytes.

See also: TBlowFishEncryptStream.Flush (129)

4.7.5 TBlowFishEncryptStream.Seek

Synopsis: Set the position in the stream

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

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 (125) exception.

See also: TBlowFishEncryptStream.Write (128)
4.7.6 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 (128), TBFBlock (125)

4.8 TBlowFishStream

4.8.1 Description

TBlowFishStream is an abstract class which is used as a parent class for TBlowFishEncryptStream (127) and TBlowFishDecryptStream (126). It simply provides a constructor and storage for a TBlowFish (125) 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 (127), TBlowFishDecryptStream (126), TBlowFish (125)

4.8.2 Method overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>129</td>
<td>Create</td>
<td>Create a new instance of the TBlowFishStream class</td>
</tr>
<tr>
<td>130</td>
<td>Destroy</td>
<td>Destroy the TBlowFishStream instance.</td>
</tr>
</tbody>
</table>

4.8.3 Property overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Properties</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>130</td>
<td>BlowFish</td>
<td>r</td>
<td>Blowfish instance used when encrypting/decrypting</td>
</tr>
</tbody>
</table>

4.8.4 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 (125) 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 (127), TBlowFishDecryptStream (126), TBlowFish (125)

### 4.8.5 TBlowFishStream.Destroy

**Synopsis:** Destroy the TBlowFishStream instance.

**Declaration:**
```cpp
destructor Destroy; Override
```

**Visibility:** public

**Description:** Destroy cleans up the internal TBlowFish (125) instance.

See also: Create (129), TBlowFish (125)

### 4.8.6 TBlowFishStream.BlowFish

**Synopsis:** Blowfish instance used when encrypting/decrypting

**Declaration:**
```cpp
Property BlowFish : TBlowFish
```

**Visibility:** public

**Access:** Read

**Description:** BlowFish is the TBlowFish (125) instance which is created when the TBlowFishStream class is initialized. Normally it should not be used directly, it’s intended for access by the descendent classes TBlowFishEncryptStream (127) and TBlowFishDecryptStream (126).

See also: TBlowFishEncryptStream (127), TBlowFishDecryptStream (126), TBlowFish (125)
Chapter 5

Reference for unit ’BufDataset’

5.1 Used units

Table 5.1: Used units by unit ’BufDataset’

<table>
<thead>
<tr>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>bufdataset_parser</td>
<td>??</td>
</tr>
<tr>
<td>Classes</td>
<td>??</td>
</tr>
<tr>
<td>DB</td>
<td>319</td>
</tr>
<tr>
<td>System</td>
<td>??</td>
</tr>
<tr>
<td>sysutils</td>
<td>??</td>
</tr>
</tbody>
</table>

5.2 Overview

Implements a database-aware buffered dataset.
Original Authors: Joost van der Sluis and members of the Free Pascal development team.

5.3 Constants, types and variables

5.3.1 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 a 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 5.2: Enumeration values for type TDataPacketFormat

<table>
<thead>
<tr>
<th>Value</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>dfAny</td>
<td>Data packets can be in any supported format (detected by the packet reader)</td>
</tr>
<tr>
<td>dfBinary</td>
<td>Data packets are in binary format</td>
</tr>
<tr>
<td>dfDefault</td>
<td>Use the default XML format. Same as using dfAny.</td>
</tr>
<tr>
<td>dfXML</td>
<td>Data packets are in XML format</td>
</tr>
<tr>
<td>dfXMLUTF8</td>
<td>Data packets are in UTF-8-encoded XML format</td>
</tr>
</tbody>
</table>

Indicates the format used for data packets in TBufDataset

TDatapacketReaderClass = Class of TDatapacketReader

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.

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 5.3: Enumeration values for type TRowStateValue

<table>
<thead>
<tr>
<th>Value</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>rsvDeleted</td>
<td>Buffer is for a record that will be deleted</td>
</tr>
<tr>
<td>rsvDetailUpdates</td>
<td>Buffer is for updates to a detail (child) dataset</td>
</tr>
<tr>
<td>rsvInserted</td>
<td>Buffer is for a record that will be inserted</td>
</tr>
<tr>
<td>rsvOriginal</td>
<td>Buffer contains the original value(s)</td>
</tr>
<tr>
<td>rsvUpdated</td>
<td>Buffer is for a record that will be updated</td>
</tr>
</tbody>
</table>

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).

5.4 Procedures and functions

5.4.1 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.
5.5 **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.

5.6 **TBufBlobField**

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

Implements a Blob field for TBufDataset

5.7 **TBufBookmark**

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

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

5.8 **TBufRecLinkItem**

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

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

5.9 **TDBCompareRec**

TDBCompareRec = record
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.

### 5.10 TRecUpdateBuffer

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

Implements a buffer used for record updates in TBufDataset

### 5.11 TArrayBufIndex

#### 5.11.1 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 (146), TRecordBuffer (131), TBufBookmark (134)
### 5.11.2 Method overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>139</td>
<td>AddRecord</td>
<td></td>
</tr>
<tr>
<td>139</td>
<td>BeginUpdate</td>
<td></td>
</tr>
<tr>
<td>138</td>
<td>CanScrollForward</td>
<td></td>
</tr>
<tr>
<td>136</td>
<td>Create</td>
<td></td>
</tr>
<tr>
<td>138</td>
<td>DoScrollForward</td>
<td></td>
</tr>
<tr>
<td>139</td>
<td>EndUpdate</td>
<td></td>
</tr>
<tr>
<td>136</td>
<td>GetCurrent</td>
<td></td>
</tr>
<tr>
<td>138</td>
<td>GotoBookmark</td>
<td></td>
</tr>
<tr>
<td>138</td>
<td>InitialiseIndex</td>
<td></td>
</tr>
<tr>
<td>138</td>
<td>InitialiseSpareRecord</td>
<td></td>
</tr>
<tr>
<td>139</td>
<td>InsertRecordBeforeCurrentRecord</td>
<td></td>
</tr>
<tr>
<td>139</td>
<td>ReleaseSpareRecord</td>
<td></td>
</tr>
<tr>
<td>139</td>
<td>RemoveRecordFromIndex</td>
<td></td>
</tr>
<tr>
<td>137</td>
<td>RestoreCurrentRecord</td>
<td></td>
</tr>
<tr>
<td>136</td>
<td>ScrollBackward</td>
<td></td>
</tr>
<tr>
<td>137</td>
<td>ScrollFirst</td>
<td></td>
</tr>
<tr>
<td>136</td>
<td>ScrollForward</td>
<td></td>
</tr>
<tr>
<td>137</td>
<td>ScrollLast</td>
<td></td>
</tr>
<tr>
<td>137</td>
<td>SetToFirstRecord</td>
<td></td>
</tr>
<tr>
<td>137</td>
<td>SetToLastRecord</td>
<td></td>
</tr>
<tr>
<td>138</td>
<td>StoreCurrentRecIntoBookmark</td>
<td></td>
</tr>
<tr>
<td>137</td>
<td>StoreCurrentRecord</td>
<td></td>
</tr>
<tr>
<td>138</td>
<td>StoreSpareRecIntoBookmark</td>
<td></td>
</tr>
</tbody>
</table>

### 5.11.3 TArrayBufIndex.Create

**Synopsis:**

**Declaration:**

```pascal
class TArrayBufIndex
  constructor Create(const ADataset: TCustomBufDataset); Override
end;
```

**Visibility:** public

### 5.11.4 TArrayBufIndex.ScrollBackward

**Synopsis:**

**Declaration:**

```pascal
function ScrollBackward : TGetResult; Override
```

**Visibility:** public

### 5.11.5 TArrayBufIndex.ScrollForward

**Synopsis:**

**Declaration:**

```pascal
function ScrollForward : TGetResult; Override
```

**Visibility:** public

### 5.11.6 TArrayBufIndex.GetCurrent

**Synopsis:**

```pascal
function GetCurrent : TGetResult; Override
```

**Visibility:** public
CHAPTER 5. REFERENCE FOR UNIT 'BUFDATASET'

Declaration: function GetCurrent : TGetResult; Override
  Visibility: public

5.11.7 TArrayBufIndex.ScrollFirst

Synopsis:

Declaration: function ScrollFirst : TGetResult; Override
  Visibility: public

5.11.8 TArrayBufIndex.ScrollLast

Synopsis:

Declaration: procedure ScrollLast; Override
  Visibility: public

5.11.9 TArrayBufIndex.SetToFirstRecord

Synopsis:

Declaration: procedure SetToFirstRecord; Override
  Visibility: public

5.11.10 TArrayBufIndex.SetToLastRecord

Synopsis:

Declaration: procedure SetToLastRecord; Override
  Visibility: public

5.11.11 TArrayBufIndex.StoreCurrentRecord

Synopsis:

Declaration: procedure StoreCurrentRecord; Override
  Visibility: public

5.11.12 TArrayBufIndex.RestoreCurrentRecord

Synopsis:

Declaration: procedure RestoreCurrentRecord; Override
  Visibility: public
5.11.13 TArrayBufIndex.CanScrollForward
Synopsis:
Declaration: function CanScrollForward : Boolean; Override
Visibility: public

5.11.14 TArrayBufIndex.DoScrollForward
Synopsis:
Declaration: procedure DoScrollForward; Override
Visibility: public

5.11.15 TArrayBufIndex.StoreCurrentRecIntoBookmark
Synopsis:
Declaration: procedure StoreCurrentRecIntoBookmark(const ABookmark: PBufBookmark)
; Override
Visibility: public

5.11.16 TArrayBufIndex.StoreSpareRecIntoBookmark
Synopsis:
Declaration: procedure StoreSpareRecIntoBookmark(const ABookmark: PBufBookmark)
; Override
Visibility: public

5.11.17 TArrayBufIndex.GotoBookmark
Synopsis:
Declaration: procedure GotoBookmark(const ABookmark: PBufBookmark); Override
Visibility: public

5.11.18 TArrayBufIndex.InitialiseIndex
Synopsis:
Declaration: procedure InitialiseIndex; Override
Visibility: public

5.11.19 TArrayBufIndex.InitialiseSpareRecord
Synopsis:
Declaration: procedure InitialiseSpareRecord(const ASpareRecord: TRecordBuffer)
; Override
Visibility: public
5.11.20  TArrayBufIndex.ReleaseSpareRecord
Synopsis:
Declaration: procedure ReleaseSpareRecord; Override
Visibility: public

5.11.21  TArrayBufIndex.BeginUpdate
Synopsis:
Declaration: procedure BeginUpdate; Override
Visibility: public

5.11.22  TArrayBufIndex.AddRecord
Synopsis:
Declaration: procedure AddRecord; Override
Visibility: public

5.11.23  TArrayBufIndex.InsertRecordBeforeCurrentRecord
Synopsis:
Declaration: procedure InsertRecordBeforeCurrentRecord(const ARecord: TRecordBuffer)
; Override
Visibility: public

5.11.24  TArrayBufIndex.RemoveRecordFromIndex
Synopsis:
Declaration: procedure RemoveRecordFromIndex(const ABookmark: TBufBookmark)
; Override
Visibility: public

5.11.25  TArrayBufIndex.EndUpdate
Synopsis:
Declaration: procedure EndUpdate; Override
Visibility: public
CHAPTER 5. REFERENCE FOR UNIT ‘BUFDATASET’

5.12 TBufBlobStream

5.12.1 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 (160)

5.12.2 Method overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>140</td>
<td>Create</td>
<td>Constructor for the class instance</td>
</tr>
<tr>
<td>140</td>
<td>Destroy</td>
<td>Destructor for the class instance</td>
</tr>
</tbody>
</table>

5.12.3 TBufBlobStream.Create

Synopsis: Constructor for the class instance

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

Visibility: public

Description: Constructor for the class instance

5.12.4 TBufBlobStream.Destroy

Synopsis: Destructor for the class instance

Declaration: destructor Destroy; Override

Visibility: public

Description: Destroy is the destructor for the class instance.

5.13 TBufDataset

5.13.1 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 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. 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 (153), TDataset (375), TDBDataset (418)

### 5.13.2 Property overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Properties</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>142</td>
<td>Active</td>
<td></td>
<td></td>
</tr>
<tr>
<td>142</td>
<td>AfterCancel</td>
<td></td>
<td></td>
</tr>
<tr>
<td>143</td>
<td>AfterClose</td>
<td></td>
<td></td>
</tr>
<tr>
<td>143</td>
<td>AfterDelete</td>
<td></td>
<td></td>
</tr>
<tr>
<td>143</td>
<td>AfterEdit</td>
<td></td>
<td></td>
</tr>
<tr>
<td>143</td>
<td>AfterInsert</td>
<td></td>
<td></td>
</tr>
<tr>
<td>143</td>
<td>AfterOpen</td>
<td></td>
<td></td>
</tr>
<tr>
<td>143</td>
<td>AfterPost</td>
<td></td>
<td></td>
</tr>
<tr>
<td>143</td>
<td>AfterScroll</td>
<td></td>
<td></td>
</tr>
<tr>
<td>142</td>
<td>AutoCalcFields</td>
<td></td>
<td></td>
</tr>
<tr>
<td>144</td>
<td>BeforeCancel</td>
<td></td>
<td></td>
</tr>
<tr>
<td>144</td>
<td>BeforeClose</td>
<td></td>
<td></td>
</tr>
<tr>
<td>144</td>
<td>BeforeDelete</td>
<td></td>
<td></td>
</tr>
<tr>
<td>144</td>
<td>BeforeEdit</td>
<td></td>
<td></td>
</tr>
<tr>
<td>144</td>
<td>BeforeInsert</td>
<td></td>
<td></td>
</tr>
<tr>
<td>144</td>
<td>BeforeOpen</td>
<td></td>
<td></td>
</tr>
<tr>
<td>144</td>
<td>BeforePost</td>
<td></td>
<td></td>
</tr>
<tr>
<td>145</td>
<td>BeforeScroll</td>
<td></td>
<td></td>
</tr>
<tr>
<td>141</td>
<td>FieldDefs</td>
<td></td>
<td></td>
</tr>
<tr>
<td>142</td>
<td>Filter</td>
<td></td>
<td></td>
</tr>
<tr>
<td>142</td>
<td>Filtered</td>
<td></td>
<td></td>
</tr>
<tr>
<td>141</td>
<td>MaxIndexesCount</td>
<td></td>
<td></td>
</tr>
<tr>
<td>145</td>
<td>OnCalcFields</td>
<td></td>
<td></td>
</tr>
<tr>
<td>145</td>
<td>OnDeleteError</td>
<td></td>
<td></td>
</tr>
<tr>
<td>145</td>
<td>OnEditError</td>
<td></td>
<td></td>
</tr>
<tr>
<td>145</td>
<td>OnFilterRecord</td>
<td></td>
<td></td>
</tr>
<tr>
<td>145</td>
<td>OnNewRecord</td>
<td></td>
<td></td>
</tr>
<tr>
<td>145</td>
<td>OnPostError</td>
<td></td>
<td></td>
</tr>
<tr>
<td>142</td>
<td>ReadOnly</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### 5.13.3 TBufDataset.MaxIndexesCount

**Declaration:**

```
Property MaxIndexesCount :
```

**Visibility:** published

**Access:**

### 5.13.4 TBufDataset.FieldDefs

**Declaration:**

```
Property FieldDefs :
```
CHAPTER 5. REFERENCE FOR UNIT 'BUF DATASET'

5.13.5 TBufDataset.Active

Declaration: Property Active :

Visibility: published
Access:

5.13.6 TBufDataset.AutoCalcFields

Declaration: Property AutoCalcFields :

Visibility: published
Access:

5.13.7 TBufDataset.Filter

Declaration: Property Filter :

Visibility: published
Access:

5.13.8 TBufDataset.Filtered

Declaration: Property Filtered :

Visibility: published
Access:

5.13.9 TBufDataset.ReadOnly

Declaration: Property ReadOnly :

Visibility: published
Access:

5.13.10 TBufDataset.AfterCancel

Declaration: Property AfterCancel :

Visibility: published
Access:
5.13.11  TBufDataset.AfterClose
Declaration: Property AfterClose :
   Visibility: published
   Access:

5.13.12  TBufDataset.AfterDelete
Declaration: Property AfterDelete :
   Visibility: published
   Access:

5.13.13  TBufDataset.AfterEdit
Declaration: Property AfterEdit :
   Visibility: published
   Access:

5.13.14  TBufDataset.AfterInsert
Declaration: Property AfterInsert :
   Visibility: published
   Access:

5.13.15  TBufDataset.AfterOpen
Declaration: Property AfterOpen :
   Visibility: published
   Access:

5.13.16  TBufDataset.AfterPost
Declaration: Property AfterPost :
   Visibility: published
   Access:

5.13.17  TBufDataset.AfterScroll
Declaration: Property AfterScroll :
   Visibility: published
   Access:
5.13.18  TBufDataset.BeforeCancel
Declaration: Property BeforeCancel :
   Visibility: published
   Access:

5.13.19  TBufDataset.BeforeClose
Declaration: Property BeforeClose :
   Visibility: published
   Access:

5.13.20  TBufDataset.BeforeDelete
Declaration: Property BeforeDelete :
   Visibility: published
   Access:

5.13.21  TBufDataset.BeforeEdit
Declaration: Property BeforeEdit :
   Visibility: published
   Access:

5.13.22  TBufDataset.BeforeInsert
Declaration: Property BeforeInsert :
   Visibility: published
   Access:

5.13.23  TBufDataset.BeforeOpen
Declaration: Property BeforeOpen :
   Visibility: published
   Access:

5.13.24  TBufDataset.BeforePost
Declaration: Property BeforePost :
   Visibility: published
   Access:
5.13.25 TBufDataset.BeforeScroll
Declaration: Property BeforeScroll :
Visibility: published
Access:

5.13.26 TBufDataset.OnCalcFields
Declaration: Property OnCalcFields :
Visibility: published
Access:

5.13.27 TBufDataset.OnDeleteError
Declaration: Property OnDeleteError :
Visibility: published
Access:

5.13.28 TBufDataset.OnEditError
Declaration: Property OnEditError :
Visibility: published
Access:

5.13.29 TBufDataset.OnFilterRecord
Declaration: Property OnFilterRecord :
Visibility: published
Access:

5.13.30 TBufDataset.OnNewRecord
Declaration: Property OnNewRecord :
Visibility: published
Access:

5.13.31 TBufDataset.OnPostError
Declaration: Property OnPostError :
Visibility: published
Access:
5.14 TBufIndex

5.14.1 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 (173), TUniDirectionalBufIndex (180), TArrayBufIndex (135)

5.14.2 Method overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>151</td>
<td>AddRecord</td>
<td>Adds a record to the index</td>
</tr>
<tr>
<td>151</td>
<td>BeginUpdate</td>
<td></td>
</tr>
<tr>
<td>150</td>
<td>BookmarkValid</td>
<td>Indicates the specified Bookmark is valid</td>
</tr>
<tr>
<td>149</td>
<td>CanScrollForward</td>
<td>Indicates if the index can be scrolled forward</td>
</tr>
<tr>
<td>150</td>
<td>CompareBookmarks</td>
<td>Gets the relative order for the specified Bookmarks</td>
</tr>
<tr>
<td>147</td>
<td>Create</td>
<td>Constructor for the class instance</td>
</tr>
<tr>
<td>149</td>
<td>DoScrollForward</td>
<td>Implements actions need to scroll forward in the index</td>
</tr>
<tr>
<td>151</td>
<td>EndUpdate</td>
<td></td>
</tr>
<tr>
<td>147</td>
<td>GetCurrent</td>
<td>Gets the record buffer for the current record in the dataset</td>
</tr>
<tr>
<td>148</td>
<td>GetRecord</td>
<td>Gets prior/next record relative to the specified bookmark</td>
</tr>
<tr>
<td>149</td>
<td>GotoBookmark</td>
<td>Moves the index position to the specified Bookmark</td>
</tr>
<tr>
<td>150</td>
<td>InitialiseIndex</td>
<td>Initializes the index</td>
</tr>
<tr>
<td>150</td>
<td>InitialiseSpareRecord</td>
<td>Initializes values in the spare record for the index</td>
</tr>
<tr>
<td>151</td>
<td>InsertRecordBeforeCurrentRecord</td>
<td>Inserts a record before the current record in the index order</td>
</tr>
<tr>
<td>151</td>
<td>OrderCurrentRecord</td>
<td></td>
</tr>
<tr>
<td>150</td>
<td>ReleaseSpareRecord</td>
<td>Releases resources allocated to the spare record for the index</td>
</tr>
<tr>
<td>151</td>
<td>RemoveRecordFromIndex</td>
<td>Remove the record at the specified bookmark from the index</td>
</tr>
<tr>
<td>149</td>
<td>RestoreCurrentRecord</td>
<td>Restores the stored record buiffer to the current record</td>
</tr>
<tr>
<td>150</td>
<td>SameBookmarks</td>
<td>Indicates if the specified Bookmarks are for the same record</td>
</tr>
<tr>
<td>147</td>
<td>ScrollBackward</td>
<td>Moves to the prior record in the index</td>
</tr>
<tr>
<td>148</td>
<td>ScrollFirst</td>
<td>Moves to the first record in the index</td>
</tr>
<tr>
<td>147</td>
<td>ScrollForward</td>
<td>Moves to the next record in the index</td>
</tr>
<tr>
<td>148</td>
<td>ScrollLast</td>
<td>Moves to the last record in the index</td>
</tr>
<tr>
<td>148</td>
<td>SetToFirstRecord</td>
<td>Sets the index to the first record in the index order</td>
</tr>
<tr>
<td>148</td>
<td>SetToLastRecord</td>
<td>Sets the index to the last record in the index order</td>
</tr>
<tr>
<td>149</td>
<td>StoreCurrentRecIntoBookmark</td>
<td>Stores the record buffer for the current record</td>
</tr>
<tr>
<td>148</td>
<td>StoreCurrentRecord</td>
<td></td>
</tr>
<tr>
<td>149</td>
<td>StoreSpareRecIntoBookmark</td>
<td></td>
</tr>
</tbody>
</table>
CHAPTER 5. REFERENCE FOR UNIT 'BUF DATASET'

5.14.3 Property overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Properties</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>153</td>
<td>BookmarkSize</td>
<td>r</td>
<td>Size (number of bytes) needed for Bookmarks in the index</td>
</tr>
<tr>
<td>152</td>
<td>CurrentBuffer</td>
<td>r</td>
<td>Current record buffer in the index</td>
</tr>
<tr>
<td>152</td>
<td>CurrentRecord</td>
<td>r</td>
<td>Current record in the index</td>
</tr>
<tr>
<td>152</td>
<td>IsInitialized</td>
<td>r</td>
<td>Indicates if the index has been initialized</td>
</tr>
<tr>
<td>153</td>
<td>RecNo</td>
<td>rw</td>
<td>Active record number in the index</td>
</tr>
<tr>
<td>152</td>
<td>SpareBuffer</td>
<td>r</td>
<td>Spare record buffer for the index</td>
</tr>
<tr>
<td>152</td>
<td>SpareRecord</td>
<td>r</td>
<td>Spare record for the index</td>
</tr>
</tbody>
</table>

5.14.4 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.

5.14.5 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 (329), TDataset.MoveBy (391)

5.14.6 TBufIndex.ScrollForward

Synopsis: Moves to the next record in the index

Declaration: function ScrollForward : TGetResult; Virtual; Abstract

Visibility: public

5.14.7 TBufIndex.GetCurrent

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

Declaration: function GetCurrent : TGetResult; Virtual; Abstract

Visibility: public
5.14.8 TBufIndex.ScrollFirst
Synopsis: Moves to the first record in the index

Declaration: function ScrollFirst : TGetResult; Virtual; Abstract
Visibility: public

5.14.9 TBufIndex.ScrollLast
Synopsis: Moves to the last record in the index

Declaration: procedure ScrollLast; Virtual; Abstract
Visibility: public

5.14.10 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.

5.14.11 TBufIndex.SetToFirstRecord
Synopsis: Sets the index to the first record in the index order

Declaration: procedure SetToFirstRecord; Virtual; Abstract
Visibility: public

5.14.12 TBufIndex.SetToLastRecord
Synopsis: Sets the index to the last record in the index order

Declaration: procedure SetToLastRecord; Virtual; Abstract
Visibility: public

5.14.13 TBufIndex.StoreCurrentRecord
Synopsis: Stores the record buffer for the current record

Declaration: procedure StoreCurrentRecord; Virtual; Abstract
Visibility: public
5.14.14  TBufIndex.RestoreCurrentRecord
Synopsis: Restores the stored record buffer to the current record
Declaration: procedure RestoreCurrentRecord; Virtual; Abstract
Visibility: public

5.14.15  TBufIndex.CanScrollForward
Synopsis: Indicates if the index can be scrolled forward
Declaration: function CanScrollForward : Boolean; Virtual; Abstract
Visibility: public

5.14.16  TBufIndex.DoScrollForward
Synopsis: Implements actions need to scroll forward in the index
Declaration: procedure DoScrollForward; Virtual; Abstract
Visibility: public

5.14.17  TBufIndex.StoreCurrentRecIntoBookmark
Synopsis:
Declaration: procedure StoreCurrentRecIntoBookmark(const ABookmark: PBufBookmark);
Visibility: public

5.14.18  TBufIndex.StoreSpareRecIntoBookmark
Synopsis:
Declaration: procedure StoreSpareRecIntoBookmark(const ABookmark: PBufBookmark);
Visibility: public

5.14.19  TBufIndex.GotoBookmark
Synopsis: Moves the index position to the specified Bookmark
Declaration: procedure GotoBookmark(const ABookmark: PBufBookmark); Virtual
Visibility: public
5.14.20 TBufIndex.BookmarkValid
Synopsis: Indicates the specified Bookmark is valid
Declaration: function BookmarkValid(const ABookmark: PBufBookmark) : Boolean
 ; Virtual
Visibility: public

5.14.21 TBufIndex.CompareBookmarks
Synopsis: Gets the relative order for the specified Bookmarks
Declaration: function CompareBookmarks(const ABookmark1: PBufBookmark;
 const ABookmark2: PBufBookmark) : Integer
 ; Virtual
Visibility: public

5.14.22 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

5.14.23 TBufIndex.InitialiseIndex
Synopsis: Initializes the index
Declaration: procedure InitialiseIndex; Virtual; Abstract
Visibility: public
Description: Initializes the index

5.14.24 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

5.14.25 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
5.14.26 TBufIndex.BeginUpdate
Synopsis:
Declaration: procedure BeginUpdate; Virtual; Abstract
Visibility: public
Description:

5.14.27 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.

5.14.28 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.

5.14.29 TBufIndex.RemoveRecordFromIndex
Synopsis: Remove the record at the specified bookmark from the index
Declaration: procedure RemoveRecordFromIndex(const ABookmark: TBufBookmark); Virtual
Visibility: public

5.14.30 TBufIndex.OrderCurrentRecord
Synopsis:
Declaration: procedure OrderCurrentRecord; Virtual; Abstract
Visibility: public
Description:

5.14.31 TBufIndex.EndUpdate
Synopsis:
Declaration: procedure EndUpdate; Virtual; Abstract
Visibility: public
Description:
5.14.32  TBufIndex.SpareRecord

Synopsis: Spare record for the index

Declaration: Property SpareRecord : TRecordBuffer
  Visibility: public
  Access: Read

Description: Spare record for the index

5.14.33  TBufIndex.SpareBuffer

Synopsis: Spare record buffer for the index

Declaration: Property SpareBuffer : TRecordBuffer
  Visibility: public
  Access: Read

Description: Spare record buffer for the index

5.14.34  TBufIndex.CurrentRecord

Synopsis: Current record in the index

Declaration: Property CurrentRecord : TRecordBuffer
  Visibility: public
  Access: Read

Description: Current record in the index

5.14.35  TBufIndex.CurrentBuffer

Synopsis: Current record buffer in the index

Declaration: Property CurrentBuffer : Pointer
  Visibility: public
  Access: Read

Description: Current record buffer in the index

5.14.36  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
5.14.37 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

5.14.38 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

5.15 TCustomBufDataset

5.15.1 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 (140), TDBDataset (418), TDataset (375)
## 5.15.2 Method overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>160</td>
<td>AddIndex</td>
<td>Adds an index definition to the dataset</td>
</tr>
<tr>
<td>156</td>
<td>ApplyUpdates</td>
<td>Applies pending updates to the dataset</td>
</tr>
<tr>
<td>165</td>
<td>BookmarkValid</td>
<td>Determines if the specified Bookmark is valid for the dataset</td>
</tr>
<tr>
<td>157</td>
<td>CancelUpdates</td>
<td>Cancels pending updates in the dataset</td>
</tr>
<tr>
<td>164</td>
<td>Clear</td>
<td>Clears the content in the dataset</td>
</tr>
<tr>
<td>161</td>
<td>ClearIndexes</td>
<td>Clears index storage in the dataset</td>
</tr>
<tr>
<td>165</td>
<td>CompareBookmarks</td>
<td>Gets the relative sort order for the specified Bookmarks</td>
</tr>
<tr>
<td>165</td>
<td>CopyFromDataset</td>
<td>Loads field definitions and optional data from another dataset</td>
</tr>
<tr>
<td>155</td>
<td>Create</td>
<td>Constructor for the class instance</td>
</tr>
<tr>
<td>160</td>
<td>CreateBlobStream</td>
<td>Creates a Blob stream for the specified field with the given permissions</td>
</tr>
<tr>
<td>164</td>
<td>CreateDataset</td>
<td>Creates the dataset using its field definitions or bound fields</td>
</tr>
<tr>
<td>157</td>
<td>Destroy</td>
<td>Destructor for the class instance</td>
</tr>
<tr>
<td>161</td>
<td>GetDatasetPacket</td>
<td>Builds a data packet representing the content in the buffered dataset</td>
</tr>
<tr>
<td>156</td>
<td>GetFieldData</td>
<td>Loads the dataset from the specified file using the given format</td>
</tr>
<tr>
<td>163</td>
<td>LoadFromFile</td>
<td>Loads the dataset from the specified file using the given data format</td>
</tr>
<tr>
<td>162</td>
<td>LoadFromStream</td>
<td>Loads the dataset from the specified stream using the given data format</td>
</tr>
<tr>
<td>158</td>
<td>Locate</td>
<td>Locates the first record with fields having the specified values</td>
</tr>
<tr>
<td>158</td>
<td>Lookup</td>
<td>Gets values from the first record with fields that match the search condition</td>
</tr>
<tr>
<td>157</td>
<td>MergeChangeLog</td>
<td>Frees update buffers and Blob update buffer allocated for the dataset</td>
</tr>
<tr>
<td>157</td>
<td>RevertRecord</td>
<td>Reverts the current record to its original (un-edited) values</td>
</tr>
<tr>
<td>163</td>
<td>SaveToFile</td>
<td>Saves the dataset to the specified file using the given data format</td>
</tr>
<tr>
<td>162</td>
<td>SaveToStream</td>
<td>Saves the dataset to the specified stream using the given data format</td>
</tr>
<tr>
<td>161</td>
<td>SetDatasetPacket</td>
<td>Retrieves and applies the data packet with the content for the dataset</td>
</tr>
<tr>
<td>156</td>
<td>SetFieldData</td>
<td></td>
</tr>
<tr>
<td>159</td>
<td>UpdateStatus</td>
<td>Gets the update status for the current record in the dataset</td>
</tr>
</tbody>
</table>
### 5.15.3 Property overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Properties</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>166</td>
<td>ChangeCount</td>
<td>r</td>
<td>Number of pending changes for the dataset</td>
</tr>
<tr>
<td>167</td>
<td>FileName</td>
<td>rw</td>
<td>File name on the local file system used to load or store the dataset</td>
</tr>
<tr>
<td>169</td>
<td>IndexDefs</td>
<td>r</td>
<td>Index definitions for the dataset</td>
</tr>
<tr>
<td>169</td>
<td>IndexFieldNames</td>
<td>rw</td>
<td>Field names included in the custom index</td>
</tr>
<tr>
<td>169</td>
<td>IndexName</td>
<td>rw</td>
<td>Name of the selected index for the dataset</td>
</tr>
<tr>
<td>167</td>
<td>ManualMergeChangeLog</td>
<td>rw</td>
<td>Indicates if the update change log can be manually merged</td>
</tr>
<tr>
<td>166</td>
<td>MaxIndexesCount</td>
<td>rw</td>
<td>Maximum number of indexes available in the dataset</td>
</tr>
<tr>
<td>168</td>
<td>OnUpdateError</td>
<td>rw</td>
<td>Event handler signalled when an error occurs while updating records</td>
</tr>
<tr>
<td>168</td>
<td>PacketRecords</td>
<td>rw</td>
<td>Number of records allowed in a data packet handled by the packet reader</td>
</tr>
<tr>
<td>167</td>
<td>ReadOnly</td>
<td>rw</td>
<td>Indicates if records can be added, deleted, or modified in the dataset</td>
</tr>
<tr>
<td>170</td>
<td>UniDirectional</td>
<td>rw</td>
<td>Indicates if the dataset is for uni-directional navigation only</td>
</tr>
</tbody>
</table>

### 5.15.4 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`)

Create 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 (166)`, `TCustomBufDataset.PacketRecords (168)`
5.15.5  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:

5.15.6  TCustomBufDataset.SetFieldData

Synopsis:

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

Visibility:  public

Description:

5.15.7  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.

5.15.8 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 (156)

5.15.9 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 (381), TDataset.Active (403), TCustomBufDataset.Resync (153)

5.15.10 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 (381), TDataset.Active (403), TCustomBufDataset.Resync (153)

5.15.11 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.

5.15.12 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 (390), TCustomBufDataset.UniDirectional (170), TDataset.Active (403), TDataset.Filter (402), TDataset.OnFilterRecord (402), TLocateOptions (330), TDBCompareStruct (132)

5.15.13 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 (158), TDataset.Active (403), TCustomBufDataset.UniDirectional (170)

5.15.14 TCustomBufDataset.UpdateStatus

Synopsis: Gets the update status for the current record in the dataset

Declaration: function UpdateStatus : TUpdateStatus; Override

Visibility: public
**CHAPTER 5. REFERENCE FOR UNIT ‘BUFDATASET’**

**Description:** `UpdateStatus` is an overridden `TUpdateStatus` function used to get the update status for the current record in the dataset. `UpdateStatus` re-implements the method defined in the ancestor class. The return value is a member of the `TUpdateStatus` enumeration, and includes the following values:

- `usUnmodified`: Record has not been modified
- `usModified`: Record has been modified
- `usInserted`: Records was appended or inserted
- `usDeleted`: Record has been deleted

`UpdateStatus` calls `GetActiveRecordUpdateBuffer` to get the update buffer for the `ActiveRecord`. When an update buffer exists for the record, its `UpdateKind` property is used as the return value for the method. The return value is `usUnmodified` when an update buffer does not exist for `ActiveRecord`.

**See also:** `TUpdateStatus` (335), `TDataset.ActiveRecord` (375)

**5.15.15 TCustomBufDataset.CreateBlobStream**

**Synopsis:** Creates a Blob stream for the specified field with the given permissions

**Declaration:**

```pascal
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` (383), `TField` (424), `TBlobStreamMode` (321)

**5.15.16 TCustomBufDataset.AddIndex**

**Synopsis:** Adds an index definition to the dataset

**Declaration:**

```pascal
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` (170), `TDataset.Active` (403), `TBufDatasetIndex` (131)
5.15.17 TCustomBufDataset.ClearIndexes

Synopsis: Clears index storage in the dataset

Declaration: procedure ClearIndexes

Visibility: public

Description: ClearIndexes is a 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 (375), TCustomBufDataset.UniDirectional (170)

5.15.18 TCustomBufDataset.SetDatasetPacket

Synopsis: Retrieves and applies the data packet with the content for the dataset

Declaration: procedure SetDatasetPacket(AReader: TDataPacketReader)

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 (392), TCustomBufDataset.LoadFromStream (162), TDataPacketReader (170)

5.15.19 TCustomBufDataset.GetDatasetPacket

Synopsis: Builds a data packet representing the content in the buffered dataset

Declaration: procedure GetDatasetPacket(AWriter: TDataPacketReader)

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 is 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 (170), TCustomBufDataset.SaveToStream (162)
CHAPTER 5. REFERENCE FOR UNIT 'BUF DATASET'

5.15.20 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 (170), TCustomBufDataset.SetDatasetPacket (161), TCustomBufDataset.LoadFromFile (163), TDataset.Open (392), TDataPacketFormat (132)

5.15.21 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 descendant 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 (170), TCustomBufDataset.SaveToFile (163), TDataPacketFormat (132), RegisterDatapacketReader (133), TDataPacketReaderClass (132), TDataPacketReader (170)
CHAPTER 5. REFERENCE FOR UNIT 'BUFDATASET'

5.15.22 TCustomBufDataset.LoadFromFile

Synopsis: Loads the dataset from the specified file using the given format

Declaration: procedure LoadFromFile(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 (167), TCustomBufDataset.LoadFromStream (162)

5.15.23 TCustomBufDataset.SaveToFile

Synopsis: Saves the dataset to the specified file using the given data format

Declaration: procedure SaveToFile(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');
CHAPTER 5. REFERENCE FOR UNIT 'BUF DATASET'

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 (167), TCustomBufDataset.SaveToStream (162)

5.15.24 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 (403), TDataSet.FieldDefs (398), TDataSet.Fields (401), TDataSet.Open (392), TCustomBufDataset.FileName (167)

5.15.25 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.


5.15.26 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 (150)

5.15.27 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 (146)

5.15.28 TCustomBufDataset.CopyFromDataset

Synopsis: Loads field definitions and optional data from another dataset

Declaration: procedure CopyFromDataset(DataSet: TDataSet; CopyData: Boolean)

Visibility: public
CHAPTER 5. REFERENCE FOR UNIT 'BUFDATASET'

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 (382), TDataset.Cancel (381), TDataset.FieldDefs (398)

5.15.29 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 TRecordUpdateBuffer 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 (156), TCustomBufDataset.CancelUpdates (157), TCustomBufDataset.RevertRecord (157)

5.15.30 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
CHAPTER 5. REFERENCE FOR UNIT ‘BUFDSATSET’

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 (403), TCustomBufDataset.AddIndex (160)

5.15.31 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 (170), TDataset.CanModify (396), TDataset.FieldDefs (398), TDataset.Fields (401)

5.15.32 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.

5.15.33 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 (163), TCustomBufDataset.SaveToFile (163), TDataset.Open (392), TDataset.Close (382)

5.15.34 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.

5.15.35 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 (133), TCustomBufDataset.ApplyUpdates (156)
5.15.36 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 (153), TIndexDefs (473)

5.15.37 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 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 (403), TDataset.Resync (393)

5.15.38 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 (153), TDataset.Active (403)

5.15.39 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 mechanisms 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 close the dataset prior to setting the value in the UniDirectional property.

See also: TDataset.ReadOnly (375), TDataset.Active (403)

5.16 TDataPacketReader

5.16.1 Description

TDataPacketReader is a TObject descendant which implements a data packet reader for TBufDataset. TDataPacketReader is a concept borrowed from TClientDataset 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).

See also: TFpcBinaryDatapacketReader (178), TXMLDatapacketReader (??)
### 5.16.2 Method overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>171</td>
<td>Create</td>
<td>Constructor for the class instance</td>
</tr>
<tr>
<td>172</td>
<td>FinalizeStoreRecords</td>
<td>Called after all records are stored</td>
</tr>
<tr>
<td>171</td>
<td>GetCurrentRecord</td>
<td>Returns True if there is at least one more record available in the stream</td>
</tr>
<tr>
<td>172</td>
<td>GetRecordRowState</td>
<td>Returns the RowState of the current record, and the order of the update</td>
</tr>
<tr>
<td>172</td>
<td>GotoNextRecord</td>
<td>Moves to the next record in the stream</td>
</tr>
<tr>
<td>171</td>
<td>InitLoadRecords</td>
<td>Called before the records are loaded</td>
</tr>
<tr>
<td>171</td>
<td>LoadFieldDefs</td>
<td>Loads the field definitions for the dataset from a stream</td>
</tr>
<tr>
<td>172</td>
<td>RecognizeStream</td>
<td>Checks if the provided stream is of the right format for this class</td>
</tr>
<tr>
<td>172</td>
<td>RestoreRecord</td>
<td>Restores a record from the stream into the current record buffer</td>
</tr>
<tr>
<td>172</td>
<td>StoreFieldDefs</td>
<td>Saves field definitions to a stream</td>
</tr>
<tr>
<td>172</td>
<td>StoreRecord</td>
<td>Saves the current record buffer to the stream</td>
</tr>
</tbody>
</table>

### 5.16.3 TDataPacketReader.Create

**Synopsis:** Constructor for the class instance

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

**Visibility:** public

### 5.16.4 TDataPacketReader.LoadFieldDefs

**Synopsis:** Loads the field definitions for the dataset from a stream

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

**Visibility:** public

### 5.16.5 TDataPacketReader.InitLoadRecords

**Synopsis:** Called before the records are loaded

**Declaration:**
```pascal
procedure InitLoadRecords; Virtual; Abstract
```

**Visibility:** public

### 5.16.6 TDataPacketReader.GetCurrentRecord

**Synopsis:** Returns True if there is at least one more record available in the stream

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

**Visibility:** public
5.16.7 **TDataPacketReader.GetRecordRowState**

Synopsis: Returns the RowState of the current record, and the order of the update

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

Visibility: public

5.16.8 **TDataPacketReader.RestoreRecord**

Synopsis: Restores a record from the stream into the current record buffer

Declaration: `procedure RestoreRecord; Virtual; Abstract`

Visibility: public

5.16.9 **TDataPacketReader.GotoNextRecord**

Synopsis: Moves to the next record in the stream

Declaration: `procedure GotoNextRecord; Virtual; Abstract`

Visibility: public

5.16.10 **TDataPacketReader.StoreFieldDefs**

Synopsis: Saves field definitions to a stream

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

Visibility: public

5.16.11 **TDataPacketReader.StoreRecord**

Synopsis: Saves the current record buffer to the stream

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

Visibility: public

5.16.12 **TDataPacketReader.FinalizeStoreRecords**

Synopsis: Called after all records are stored

Declaration: `procedure FinalizeStoreRecords; Virtual; Abstract`

Visibility: public

5.16.13 **TDataPacketReader.RecognizeStream**

Synopsis: Checks if the provided stream is of the right format for this class

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

Visibility: public
5.17 TDoubleLinkedBufIndex

5.17.1 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 (146), TBufRecLinkItem (134), PBufRecLinkItem (132), TBufDataset (140)

5.17.2 Method overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>177</td>
<td>AddRecord</td>
<td>Adds a record to the index</td>
</tr>
<tr>
<td>177</td>
<td>BeginUpdate</td>
<td></td>
</tr>
<tr>
<td>175</td>
<td>CanScrollForward</td>
<td>Indicates if the index can be scrolled towards the end of the index</td>
</tr>
<tr>
<td>176</td>
<td>CompareBookmarks</td>
<td></td>
</tr>
<tr>
<td>175</td>
<td>DoScrollForward</td>
<td>Implements actions needed to scroll forward using the index</td>
</tr>
<tr>
<td>177</td>
<td>EndUpdate</td>
<td></td>
</tr>
<tr>
<td>174</td>
<td>GetCurrent</td>
<td></td>
</tr>
<tr>
<td>174</td>
<td>GetRecord</td>
<td></td>
</tr>
<tr>
<td>176</td>
<td>GotoBookmark</td>
<td>Navigates the index to the specified Bookmark</td>
</tr>
<tr>
<td>176</td>
<td>InitialiseIndex</td>
<td>Initializes the index</td>
</tr>
<tr>
<td>176</td>
<td>InitialiseSpareRecord</td>
<td>Initializes the spare record in the index</td>
</tr>
<tr>
<td>177</td>
<td>InsertRecordBeforeCurrentRecord</td>
<td>Inserts a record prior to the current record in the index</td>
</tr>
<tr>
<td>177</td>
<td>OrderCurrentRecord</td>
<td></td>
</tr>
<tr>
<td>176</td>
<td>ReleaseSpareRecord</td>
<td>Releases the spare record in the index</td>
</tr>
<tr>
<td>177</td>
<td>RemoveRecordFromIndex</td>
<td>Removes the record at the specified Bookmark from the index</td>
</tr>
<tr>
<td>175</td>
<td>RestoreCurrentRecord</td>
<td></td>
</tr>
<tr>
<td>176</td>
<td>SameBookmarks</td>
<td>Compares Bookmark values for ordering in the index</td>
</tr>
<tr>
<td>173</td>
<td>ScrollBackward</td>
<td>Scrolls the index toward the beginning of the index</td>
</tr>
<tr>
<td>174</td>
<td>ScrollFirst</td>
<td>Scrolls to the first entry in the index</td>
</tr>
<tr>
<td>174</td>
<td>ScrollForward</td>
<td>Scrolls the index towards the end of the index</td>
</tr>
<tr>
<td>174</td>
<td>ScrollLast</td>
<td>Scrolls to the last entry in the index</td>
</tr>
<tr>
<td>174</td>
<td>SetToFirstRecord</td>
<td></td>
</tr>
<tr>
<td>175</td>
<td>SetToLastRecord</td>
<td></td>
</tr>
<tr>
<td>175</td>
<td>StoreCurrentRecIntoBookmark</td>
<td></td>
</tr>
<tr>
<td>175</td>
<td>StoreCurrentRecord</td>
<td></td>
</tr>
<tr>
<td>175</td>
<td>StoreSpareRecIntoBookmark</td>
<td></td>
</tr>
</tbody>
</table>

5.17.3 TDoubleLinkedBufIndex.ScrollBackward

Synopsis: Scrolls the index toward the beginning of the index

Declaration: function ScrollBackward : TGetResult; Override
5.17.4 TDoubleLinkedBufIndex.ScrollForward
Synopsis: Scrolls the index towards the end of the index
Declaration: function ScrollForward: TGetResult; Override
  Visibility: public

5.17.5 TDoubleLinkedBufIndex.GetCurrent
Synopsis:
Declaration: function GetCurrent: TGetResult; Override
  Visibility: public

5.17.6 TDoubleLinkedBufIndex.ScrollFirst
Synopsis: Scrolls to the first entry in the index
Declaration: function ScrollFirst: TGetResult; Override
  Visibility: public

5.17.7 TDoubleLinkedBufIndex.ScrollLast
Synopsis: Scrolls to the last entry in the index
Declaration: procedure ScrollLast; Override
  Visibility: public

5.17.8 TDoubleLinkedBufIndex.GetRecord
Synopsis:
Declaration: function GetRecord(ABookmark: PBufBookmark; GetMode: TGetMode) : TGetResult; Override
  Visibility: public

5.17.9 TDoubleLinkedBufIndex.SetToFirstRecord
Synopsis:
Declaration: procedure SetToFirstRecord; Override
  Visibility: public
5.17.10  TDoubleLinkedBufIndex.SetToLastRecord
Synopsis:
Declaration: procedure SetToLastRecord; Override
Visibility: public

5.17.11  TDoubleLinkedBufIndex.StoreCurrentRecord
Synopsis:
Declaration: procedure StoreCurrentRecord; Override
Visibility: public

5.17.12  TDoubleLinkedBufIndex.RestoreCurrentRecord
Synopsis:
Declaration: procedure RestoreCurrentRecord; Override
Visibility: public

5.17.13  TDoubleLinkedBufIndex.CanScrollForward
Synopsis: Indicates if the index can be scrolled towards the end of the index
Declaration: function CanScrollForward : Boolean; Override
Visibility: public

5.17.14  TDoubleLinkedBufIndex.DoScrollForward
Synopsis: Implements actions needed to scroll forward using the index
Declaration: procedure DoScrollForward; Override
Visibility: public

5.17.15  TDoubleLinkedBufIndex.StoreCurrentRecIntoBookmark
Synopsis:
Declaration: procedure StoreCurrentRecIntoBookmark(const ABookmark: PBufBookmark);
Visibility: public

5.17.16  TDoubleLinkedBufIndex.StoreSpareRecIntoBookmark
Synopsis:
Declaration: procedure StoreSpareRecIntoBookmark(const ABookmark: PBufBookmark);
Visibility: public
5.17.17  TDoubleLinkedBufIndex.GotoBookmark
Synopsis: Navigates the index to the specified Bookmark
Declaration: procedure GotoBookmark(const ABookmark: PBufBookmark); Override
Visibility: public

5.17.18  TDoubleLinkedBufIndex.CompareBookmarks
Synopsis:
Declaration: function CompareBookmarks(const ABookmark1: PBufBookmark;
 const ABookmark2: PBufBookmark) : Integer ; Override
Visibility: public

5.17.19  TDoubleLinkedBufIndex.SameBookmarks
Synopsis: Compares Bookmark values for ordering in the index
Declaration: function SameBookmarks(const ABookmark1: PBufBookmark;
 const ABookmark2: PBufBookmark) : Boolean ; Override
Visibility: public

5.17.20  TDoubleLinkedBufIndex.InitialiseIndex
Synopsis: Initializes the index
Declaration: procedure InitialiseIndex; Override
Visibility: public

5.17.21  TDoubleLinkedBufIndex.InitialiseSpareRecord
Synopsis: Initializes the spare record in the index
Declaration: procedure InitialiseSpareRecord(const ASpareRecord: TRecordBuffer) ; Override
Visibility: public

5.17.22  TDoubleLinkedBufIndex.ReleaseSpareRecord
Synopsis: Releases the spare record in the index
Declaration: procedure ReleaseSpareRecord; Override
Visibility: public
5.17.23  **TDoubleLinkedBufIndex.BeginUpdate**

Synopsis:

**Declaration:** procedure BeginUpdate; Override

**Visibility:** public

5.17.24  **TDoubleLinkedBufIndex.AddRecord**

Synopsis: Adds a record to the index

**Declaration:** procedure AddRecord; Override

**Visibility:** public

5.17.25  **TDoubleLinkedBufIndex.InsertRecordBeforeCurrentRecord**

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

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

**Visibility:** public

5.17.26  **TDoubleLinkedBufIndex.RemoveRecordFromIndex**

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

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

**Visibility:** public

5.17.27  **TDoubleLinkedBufIndex.OrderCurrentRecord**

Synopsis:

**Declaration:** procedure OrderCurrentRecord; Override

**Visibility:** public

5.17.28  **TDoubleLinkedBufIndex.EndUpdate**

Synopsis:

**Declaration:** procedure EndUpdate; Override

**Visibility:** public
5.18 TFpcBinaryDatapacketReader

5.18.1 Description

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

Table 5.4:

<table>
<thead>
<tr>
<th>Section</th>
<th>Name</th>
<th>Length</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Header</td>
<td>Identification</td>
<td>13 bytes</td>
<td>Contains the value 'BinBufDataSet'</td>
</tr>
<tr>
<td></td>
<td>Version</td>
<td>1 byte</td>
<td>Version number</td>
</tr>
<tr>
<td></td>
<td>Field Definition Count</td>
<td>2 bytes</td>
<td>Column Layout for field Section occurs the number of times indicated in Field Definition Count</td>
</tr>
<tr>
<td>Field Definitions</td>
<td>Field Name Length</td>
<td>2 bytes</td>
<td>Name of the field in the dataset</td>
</tr>
<tr>
<td></td>
<td>Field Name</td>
<td>See above</td>
<td>Caption for the field</td>
</tr>
<tr>
<td></td>
<td>Display Name Length</td>
<td>2 bytes</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Display Name</td>
<td>See above</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Data Type</td>
<td>2 bytes</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Read-Only Attribute</td>
<td>1 byte</td>
<td>0=read/write, 1=read-only</td>
</tr>
<tr>
<td>Parameters</td>
<td>AutoInc Value</td>
<td>4 bytes</td>
<td>Integer value for the auto-increment field type</td>
</tr>
<tr>
<td></td>
<td>Row Marker</td>
<td>1 byte</td>
<td>Contains the value $fe</td>
</tr>
<tr>
<td></td>
<td>Row State</td>
<td>1 byte</td>
<td>0=original, 1=deleted</td>
</tr>
<tr>
<td></td>
<td>Update Order</td>
<td>4 bytes</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Null Bitmap</td>
<td>Variable length.</td>
<td>1 byte required for eight Variable-length fields a</td>
</tr>
<tr>
<td>Row Header</td>
<td>Optional Field Length</td>
<td>4 bytes</td>
<td>Occurs the number of</td>
</tr>
<tr>
<td></td>
<td>Field Data</td>
<td>Bytes values for the field. See Optional Field Length.</td>
<td></td>
</tr>
</tbody>
</table>

See also: TDatapacketReader (170)

5.18.2 Method overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>178</td>
<td>Create</td>
<td>Constructor for the class instance</td>
</tr>
<tr>
<td>180</td>
<td>FinalizeStoreRecords</td>
<td></td>
</tr>
<tr>
<td>179</td>
<td>GetCurrentRecord</td>
<td></td>
</tr>
<tr>
<td>179</td>
<td>GetRecordRowState</td>
<td></td>
</tr>
<tr>
<td>179</td>
<td>GotoNextRecord</td>
<td></td>
</tr>
<tr>
<td>179</td>
<td>InitLoadRecords</td>
<td></td>
</tr>
<tr>
<td>179</td>
<td>LoadFieldDefs</td>
<td>Loads field definitions using the data packet reader</td>
</tr>
<tr>
<td>180</td>
<td>RecognizeStream</td>
<td></td>
</tr>
<tr>
<td>179</td>
<td>RestoreRecord</td>
<td></td>
</tr>
<tr>
<td>179</td>
<td>StoreFieldDefs</td>
<td>Stores field definitions using the data packet reader</td>
</tr>
<tr>
<td>180</td>
<td>StoreRecord</td>
<td></td>
</tr>
</tbody>
</table>

5.18.3 TFpcBinaryDatapacketReader.Create

Synopsis: Constructor for the class instance

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

Visibility: public
5.18.4 TFpcBinaryDatapacketReader.LoadFieldDefs
Synopsis: Loads field definitions using the data packet reader
Declaration: procedure LoadFieldDefs(var AnAutoIncValue: Integer); Override
Visibility: public

5.18.5 TFpcBinaryDatapacketReader.StoreFieldDefs
Synopsis: Stores field definitions using the data packet reader
Declaration: procedure StoreFieldDefs(AnAutoIncValue: Integer); Override
Visibility: public

5.18.6 TFpcBinaryDatapacketReader.InitLoadRecords
Synopsis:
Declaration: procedure InitLoadRecords; Override
Visibility: public

5.18.7 TFpcBinaryDatapacketReader.GetCurrentRecord
Synopsis:
Declaration: function GetCurrentRecord : Boolean; Override
Visibility: public

5.18.8 TFpcBinaryDatapacketReader.GetRecordRowState
Synopsis:
Declaration: function GetRecordRowState(out AUpdOrder: Integer) : TRowState ; Override
Visibility: public

5.18.9 TFpcBinaryDatapacketReader.RestoreRecord
Synopsis:
Declaration: procedure RestoreRecord; Override
Visibility: public

5.18.10 TFpcBinaryDatapacketReader.GotoNextRecord
Synopsis:
Declaration: procedure GotoNextRecord; Override
Visibility: public
5.18.11 TFpcBinaryDatapacketReader.StoreRecord

Synopsis:

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

Visibility: public

5.18.12 TFpcBinaryDatapacketReader.FinalizeStoreRecords

Synopsis:

Declaration: procedure FinalizeStoreRecords; Override

Visibility: public

5.18.13 TFpcBinaryDatapacketReader.RecognizeStream

Synopsis:

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

Visibility: public

5.19 TUniDirectionalBufIndex

5.19.1 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 (146)
### 5.19.2 Method overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>184</td>
<td>AddRecord</td>
<td></td>
</tr>
<tr>
<td>184</td>
<td>BeginUpdate</td>
<td></td>
</tr>
<tr>
<td>182</td>
<td>CanScrollForward</td>
<td></td>
</tr>
<tr>
<td>183</td>
<td>DoScrollForward</td>
<td></td>
</tr>
<tr>
<td>184</td>
<td>EndUpdate</td>
<td></td>
</tr>
<tr>
<td>181</td>
<td>GetCurrent</td>
<td></td>
</tr>
<tr>
<td>183</td>
<td>GotoBookmark</td>
<td></td>
</tr>
<tr>
<td>183</td>
<td>InitialiseIndex</td>
<td></td>
</tr>
<tr>
<td>183</td>
<td>InitialiseSpareRecord</td>
<td></td>
</tr>
<tr>
<td>184</td>
<td>InsertRecordBeforeCurrentRecord</td>
<td></td>
</tr>
<tr>
<td>184</td>
<td>OrderCurrentRecord</td>
<td></td>
</tr>
<tr>
<td>183</td>
<td>ReleaseSpareRecord</td>
<td></td>
</tr>
<tr>
<td>184</td>
<td>RemoveRecordFromIndex</td>
<td></td>
</tr>
<tr>
<td>182</td>
<td>RestoreCurrentRecord</td>
<td></td>
</tr>
<tr>
<td>181</td>
<td>ScrollBackward</td>
<td></td>
</tr>
<tr>
<td>181</td>
<td>ScrollFirst</td>
<td></td>
</tr>
<tr>
<td>181</td>
<td>ScrollForward</td>
<td></td>
</tr>
<tr>
<td>182</td>
<td>ScrollLast</td>
<td></td>
</tr>
<tr>
<td>182</td>
<td>SetToFirstRecord</td>
<td></td>
</tr>
<tr>
<td>182</td>
<td>SetToLastRecord</td>
<td></td>
</tr>
<tr>
<td>183</td>
<td>StoreCurrentRecIntoBookmark</td>
<td></td>
</tr>
<tr>
<td>182</td>
<td>StoreCurrentRecord</td>
<td></td>
</tr>
<tr>
<td>183</td>
<td>StoreSpareRecIntoBookmark</td>
<td></td>
</tr>
</tbody>
</table>

### 5.19.3 TUniDirectionalBufIndex.ScrollBackward

**Synopsis:**

**Declaration:** `function ScrollBackward : TGetResult; Override`

**Visibility:** public

### 5.19.4 TUniDirectionalBufIndex.ScrollForward

**Synopsis:**

**Declaration:** `function ScrollForward : TGetResult; Override`

**Visibility:** public

### 5.19.5 TUniDirectionalBufIndex.GetCurrent

**Synopsis:**

**Declaration:** `function GetCurrent : TGetResult; Override`

**Visibility:** public

### 5.19.6 TUniDirectionalBufIndex.ScrollFirst

**Synopsis:**
Declaration: function ScrollFirst : TGetResult; Override
Visibility: public

5.19.7 TUniDirectionalBufIndex.ScrollLast
Synopsis:
Declaration: procedure ScrollLast; Override
Visibility: public

5.19.8 TUniDirectionalBufIndex.SetToFirstRecord
Synopsis:
Declaration: procedure SetToFirstRecord; Override
Visibility: public

5.19.9 TUniDirectionalBufIndex.SetToLastRecord
Synopsis:
Declaration: procedure SetToLastRecord; Override
Visibility: public

5.19.10 TUniDirectionalBufIndex.StoreCurrentRecord
Synopsis:
Declaration: procedure StoreCurrentRecord; Override
Visibility: public

5.19.11 TUniDirectionalBufIndex.RestoreCurrentRecord
Synopsis:
Declaration: procedure RestoreCurrentRecord; Override
Visibility: public

5.19.12 TUniDirectionalBufIndex.CanScrollForward
Synopsis:
Declaration: function CanScrollForward : Boolean; Override
Visibility: public
5.19.13  TUniDirectionalBufIndex.DoScrollForward
Synopsis:
Declaration: procedure DoScrollForward; Override
Visibility: public

5.19.14  TUniDirectionalBufIndex.StoreCurrentRecIntoBookmark
Synopsis:
Declaration: procedure StoreCurrentRecIntoBookmark(const ABookmark: PBufBookmark);
Visibility: public

5.19.15  TUniDirectionalBufIndex.StoreSpareRecIntoBookmark
Synopsis:
Declaration: procedure StoreSpareRecIntoBookmark(const ABookmark: PBufBookmark);
Visibility: public

5.19.16  TUniDirectionalBufIndex.GotoBookmark
Synopsis:
Declaration: procedure GotoBookmark(const ABookmark: PBufBookmark); Override
Visibility: public

5.19.17  TUniDirectionalBufIndex.InitialiseIndex
Synopsis:
Declaration: procedure InitialiseIndex; Override
Visibility: public

5.19.18  TUniDirectionalBufIndex.InitialiseSpareRecord
Synopsis:
Declaration: procedure InitialiseSpareRecord(const ASpareRecord: TRecordBuffer);
Visibility: public

5.19.19  TUniDirectionalBufIndex.ReleaseSpareRecord
Synopsis:
Declaration: procedure ReleaseSpareRecord; Override
Visibility: public
5.19.20  TUniDirectionalBufIndex.BeginUpdate

Synopsis:

Declaration: procedure BeginUpdate; Override

Visibility: public

5.19.21  TUniDirectionalBufIndex.AddRecord

Synopsis:

Declaration: procedure AddRecord; Override

Visibility: public

5.19.22  TUniDirectionalBufIndex.InsertRecordBeforeCurrentRecord

Synopsis:

Declaration: procedure InsertRecordBeforeCurrentRecord(const ARecord: TRecordBuffer)

; Override

Visibility: public

5.19.23  TUniDirectionalBufIndex.RemoveRecordFromIndex

Synopsis:

Declaration: procedure RemoveRecordFromIndex(const ABookmark: TBufBookmark)

; Override

Visibility: public

5.19.24  TUniDirectionalBufIndex.OrderCurrentRecord

Synopsis:

Declaration: procedure OrderCurrentRecord; Override

Visibility: public

5.19.25  TUniDirectionalBufIndex.EndUpdate

Synopsis:

Declaration: procedure EndUpdate; Override

Visibility: public
Chapter 6

Reference for unit ’bufstream’

6.1 Used units

<table>
<thead>
<tr>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Classes</td>
<td>??</td>
</tr>
<tr>
<td>System</td>
<td>??</td>
</tr>
<tr>
<td>sysutils</td>
<td>??</td>
</tr>
</tbody>
</table>

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

TWriteBufStream (192) is used for writing only, and allows the buffer size to be specified at the time of creation.

TBufferedFileStream (186) 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.

6.3 Constants, types and variables

6.3.1 Constants

DefaultBufferCapacity : Integer = 16

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

6.4.1 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 4,096 bytes per page. Both page count and page size configurable using methods 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, when the Flush method is called, and when the class instance is freed.

See also: TFileStream (?), THandleStream (?), TStream (?)

6.4.2 Method overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>186</td>
<td>Create</td>
<td>Constructor for the class instance</td>
</tr>
<tr>
<td>187</td>
<td>Destroy</td>
<td>Destructor for the class instance</td>
</tr>
<tr>
<td>189</td>
<td>Flush</td>
<td>Flushes modified pages in the buffer to the file stream</td>
</tr>
<tr>
<td>189</td>
<td>InitializeCache</td>
<td>Re-initializes the internal buffer for the buffered file stream</td>
</tr>
<tr>
<td>188</td>
<td>Read</td>
<td>Reads the specified number of bytes into the Buffer parameter</td>
</tr>
<tr>
<td>187</td>
<td>Seek</td>
<td>Moves the position in the buffer relative to the specified origin</td>
</tr>
<tr>
<td>188</td>
<td>Write</td>
<td>Writes the specified number of bytes in Buffer to the internal page buffer(s)</td>
</tr>
</tbody>
</table>

6.4.3 TBufferedFileStream.Create

Synopsis: Constructor for the class instance

Declaration: constructor Create(const AFileName: string; Mode: Word)
             constructor Create(const AFileName: string; Mode: Word;
                                  Rights: Cardinal)

Visibility: public

Description: Create is the constructor for the class instance. Overloaded variants are provided to match the constructors used in the ancestor class (TFileStream).

Create ensures that resources are allocated for the internal buffer. By default, the buffer reserves 8 blocks (pages) with 4,096 bytes per block (page). Create calls InitializeCache to allocate resources needed for the internal buffer.

Create calls the inherited constructor using the parameter values passed to the method.

AFileName is the qualified path to the file where the content in the stream is stored.

Mode contains the file mode used for the file handle in the ancestor class. It uses the following file mode constant values:

- fmCreate Creates the file it does not already exist.
- fmOpenRead Opens the file for read-only access.
- fmOpenWrite Opens the file for write-only access.
- fmOpenReadWrite Opens the file for read / write access.

The file mode constants (except for fmCreate) can be OR’d with sharing mode constants, including:
fmShareCompat Opens the file in DOS-compatibility sharing mode.

fmShareExclusive Locks the file for exclusive use.

fmShareDenyWrite Locks the file and denies write access to other processes.

fmShareDenyRead Locks the file and denies read access to other processes.

fmShareDenyNone Does not lock the file.

Rights contains the value used as the file mode on UNIX-like file systems. It contains a value representing the read, write, execute, sticky-bit, setgid, and setuid flags used on the platform. It is ignored for all other platforms, and is significant only when using fmCreate in Mode. The Size for the internal buffer is updated to use the length of the file stream.

See also: TBufferedFileStream.InitializeCache (189), TBufferedFileStream.Size (186), TFileStream.Create (??), TFileStream.Size (??)

6.4.4 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 (??)

6.4.5 TBufferedFileStream.Seek

Synopsis: Moves the position in the buffer relative to the specified origin

Declaration: function Seek(Offset: LongInt; Origin: Word) : LongInt; Override;
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 (??)
6.4.6 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 of directly accessing the underlying file stream. It locates the page in the buffer corresponding to the current 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 stream buffer 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 (??)

6.4.7 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 corresponding to the current 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.

6.4.8 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 (188), THandleStream.Seek (?), THandleStream.Write (?), TStream.Seek (?), TStream.Write (?), EStreamError (?)

6.4.9 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 (189), TBufferedFileStream.Create (186)

6.5 TBufStream

6.5.1 Description

TBufStream is the common ancestor for the TReadBufStream (191) and TWriteBufStream (192) 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 (191), TWriteBufStream (192)
6.5.2 Method overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>190</td>
<td>Create</td>
<td>Create a new TBufStream instance.</td>
</tr>
<tr>
<td>190</td>
<td>Destroy</td>
<td>Destroys the TBufStream instance</td>
</tr>
</tbody>
</table>

6.5.3 Property overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Properties</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>190</td>
<td>Buffer</td>
<td>r</td>
<td>The current buffer</td>
</tr>
<tr>
<td>191</td>
<td>BufferPos</td>
<td>r</td>
<td>Current buffer position.</td>
</tr>
<tr>
<td>191</td>
<td>BufferSize</td>
<td>r</td>
<td>Amount of data in the buffer</td>
</tr>
<tr>
<td>191</td>
<td>Capacity</td>
<td>rw</td>
<td>Current buffer capacity</td>
</tr>
</tbody>
</table>

6.5.4 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 (185) is used as the capacity.

An instance of TBufStream should never be instantiated directly. Instead, an instance of TReadBufStream (191) or TWriteBufStream (192) should be created.

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

See also: TBufStream.Destroy (190), TReadBufStream (191), TWriteBufStream (192)

6.5.5 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 (190), TReadBufStream (191), TWriteBufStream (192)

6.5.6 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 (190), TBufStream.Capacity (191), TBufStream.BufferSize (191)
6.5.7 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.


6.5.8 TBufStream.BufferPos

Synopsis: Current buffer position.

Declaration: Property BufferPos : Integer

Visibility: public

Access: Read

Description: BufferPos 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.


6.5.9 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 (191).

See also: TBufStream.Create (190), TBufStream.Buffer (190), TBufStream.BufferPos (191), TBufStream.Capacity (191)

6.6 TReadBufStream

6.6.1 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 (189), TWriteBufStream (192)
6.6.2 Method overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>192</td>
<td>Read</td>
<td>Reads data from the stream</td>
</tr>
<tr>
<td>192</td>
<td>Seek</td>
<td>Set location in the buffer</td>
</tr>
</tbody>
</table>

6.6.3 TReadBufStream.Seek

Synopsis: Set location in the buffer

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

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 (193), TReadBufStream.Read (192)

6.6.4 TReadBufStream.Read

Synopsis: Reads data from the stream

Declaration: function Read(var ABuffer; ACount: LongInt) : Integer;

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 (192), TReadBufStream.Read (192)

6.7 TWriteBufStream

6.7.1 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 (189), TReadBufStream (191)

6.7.2 Method overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>193</td>
<td>Destroy</td>
<td>Remove the TWriteBufStream instance from memory</td>
</tr>
<tr>
<td>193</td>
<td>Seek</td>
<td>Set stream position</td>
</tr>
<tr>
<td>193</td>
<td>Write</td>
<td>Write data to the stream</td>
</tr>
</tbody>
</table>
6.7.3 TWriteBufStream.Destroy

**Synopsis:** Remove the TWriteBufStream instance from memory

**Declaration:**
```plaintext
destructor Destroy; Override
```

**Visibility:** public

**Description:**
Destroy flushes the buffer and then calls the inherited Destroy (190).

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

**See also:** Create (190), TBufStream.Destroy (190)

6.7.4 TWriteBufStream.Seek

**Synopsis:** Set stream position.

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

**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 (193), TReadBufStream.Seek (192)

6.7.5 TWriteBufStream.Write

**Synopsis:** Write data to the stream

**Declaration:**
```plaintext
function Write(const ABuffer; ACount: LongInt) : Integer;
```

**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 (193)
Chapter 7

Reference for unit ’CacheCls’

7.1 Used units

Table 7.1: Used units by unit ’CacheCls’

<table>
<thead>
<tr>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>System</td>
<td>??</td>
</tr>
<tr>
<td>sysutils</td>
<td>??</td>
</tr>
</tbody>
</table>

7.2 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

7.3 Constants, types and variables

7.3.1 Resource strings

SInvalidIndex = 'Invalid index %i'

Message shown when an invalid index is passed.

7.3.2 Types

PCacheSlot = ^TCacheSlot

Pointer to TCacheSlot (195) record.

PCacheSlotArray = ^TCacheSlotArray

Pointer to TCacheSlotArray (195) array

TCacheSlotArray = Array[0..MaxIntdivSizeOf(TCacheSlot)-1] of TCacheSlot
TCacheSlotArray is an array of TCacheSlot items. Do not use TCacheSlotArray directly, instead, use PCacheSlotArray (194) 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 (196) 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 anisstrings, then the contents should be compared.

7.4 TCacheSlot

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

TCacheSlot is internally used by the TCache (195) class. It represents 1 element in the linked list.

7.5 ECacheError

7.5.1 Description

Exception class used in the cachecls unit.

7.6 TCache

7.6.1 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 (198) property offers indexed access to the array of items. When accessing the array through this property, the MRUSlot (198) property is updated.
CHAPTER 7. REFERENCE FOR UNIT ‘CACHECLS’

7.6.2 Method overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>196</td>
<td>Add</td>
<td>Add a data element to the list.</td>
</tr>
<tr>
<td>197</td>
<td>AddNew</td>
<td>Add a new item to the list.</td>
</tr>
<tr>
<td>196</td>
<td>Create</td>
<td>Create a new cache class.</td>
</tr>
<tr>
<td>196</td>
<td>Destroy</td>
<td>Free the TCache class from memory</td>
</tr>
<tr>
<td>197</td>
<td>FindSlot</td>
<td>Find data pointer in the list</td>
</tr>
<tr>
<td>197</td>
<td>IndexOf</td>
<td>Return index of a data pointer in the list</td>
</tr>
<tr>
<td>198</td>
<td>Remove</td>
<td>Remove a data item from the list</td>
</tr>
</tbody>
</table>

7.6.3 Property overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Properties</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>198</td>
<td>Data</td>
<td>rw</td>
<td>Indexed access to data items</td>
</tr>
<tr>
<td>199</td>
<td>LRUSlot</td>
<td>r</td>
<td>Last used item</td>
</tr>
<tr>
<td>198</td>
<td>MRUSlot</td>
<td>rw</td>
<td>Most recent item slot.</td>
</tr>
<tr>
<td>200</td>
<td>OnFreeSlot</td>
<td>rw</td>
<td>Event called when a slot is freed</td>
</tr>
<tr>
<td>199</td>
<td>OnIsDataEqual</td>
<td>rw</td>
<td>Event to compare 2 items.</td>
</tr>
<tr>
<td>199</td>
<td>SlotCount</td>
<td>rw</td>
<td>Number of slots in the list</td>
</tr>
<tr>
<td>199</td>
<td>Slots</td>
<td>r</td>
<td>Indexed array to the slots</td>
</tr>
</tbody>
</table>

7.6.4 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 (199)

7.6.5 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 (196)

7.6.6 TCache.Add

Synopsis: Add a data element to the list.

Declaration: function Add(AData: Pointer) : Integer

Visibility: public
CHAPTER 7. REFERENCE FOR UNIT ‘CACHECLS’

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 (197) 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 (197), TCache.FindSlot (197), TCache.IndexOf (197), TCache.Data (198), TCache.MRUSlot (198)

7.6.7 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 (196) 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 (196), TCache.FindSlot (197), TCache.IndexOf (197), TCache.Data (198), TCache.MRUSlot (198)

7.6.8 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 (199) event must be set.

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

See also: TCache.IndexOf (197), TCache.Add (196), TCache.OnIsDataEqual (199)

7.6.9 TCache.IndexOf

Synopsis: Return index of a data pointer in the list.

Declaration: functionIndexOf(AData: Pointer) : Integer

Visibility: public
CHAPTER 7. REFERENCE FOR UNIT ‘CACHECLS’

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 (199) event must be set.

Errors: If OnIsDataEqual is not set, an exception will be raised.
See also: TCache.FindSlot (197), TCache.Add (196), TCache.OnIsDataEqual (199)

7.6.10 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 (197)

7.6.11 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 (198) will point to the accessed item. The access is both read and write.
The index is zero-based and can maximally be SlotCount-1 (199). Providing an invalid index will result in an exception.
See also: TCache.MRUSlot (198)

7.6.12 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 (198) property, or when an item is added to the list with Add (196) or AddNew (197)
See also: TCache.Add (196), TCache.AddNew (197), TCache.Data (198), TCache.LRUSlot (199)
7.6.13 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 (196), TCache.AddNew (197), TCache.Data (198), TCache.MRUSlot (198)

7.6.14 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 (200) is called.

See also: TCache.Create (196), TCache.Data (198), TCache.Slots (199)

7.6.15 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 (199). Providing an invalid index will result in an exception.

See also: TCache.Data (198), TCache.SlotCount (199)

7.6.16 TCache.OnIsDataEqual
Synopsis: Event to compare 2 items.

Declaration: Property OnIsDataEqual : TOnIsDataEqual
   Visibility: public
   Access: Read, Write
Description: OnIsDataEqual is used by FindSlot (197) and IndexOf (197) to compare items when looking for a particular item. These functions are called by the Add (196) 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 (197), TCache.IndexOf (197), TCache.Add (196)

7.6.17 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 (196), TCache.AddNew (197), TCache.SlotCount (199)
Chapter 8

Reference for unit ’Contnrs’

8.1 Used units

Table 8.1: Used units by unit ’Contnrs’

<table>
<thead>
<tr>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Classes</td>
<td>??</td>
</tr>
<tr>
<td>System</td>
<td>??</td>
</tr>
<tr>
<td>sysutils</td>
<td>??</td>
</tr>
</tbody>
</table>

8.2 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.

8.3 Constants, types and variables

8.3.1 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.

### 8.3.2 Types

PBucket = ^TBucket

Pointer to TBucket (205)" type.

PHashItem = ^THashItem

PHashItem is a pointer type, pointing to the THashItem (206) record.

PHashItemList = ^THashItemList

PHashItemList is a pointer to the THashItemList (203). It’s used in the TFPHashList (222) as a pointer to the memory area containing the hash item records.

PHashTable = ^THashTable

PHashTable is a pointer to the THashTable (203). It’s used in the TFPHashList (222) as a pointer to the memory area containing the hash values.

TBucketArray = Array of TBucket

Array of TBucket (205) records.

TBucketItemArray = Array of TBucketItem

Array of TBucketItem records

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

Table 8.2: Enumeration values for type TBucketListSizes

<table>
<thead>
<tr>
<th>Value</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>bl128</td>
<td>List with 128 buckets</td>
</tr>
<tr>
<td>bl16</td>
<td>List with 16 buckets</td>
</tr>
<tr>
<td>bl2</td>
<td>List with 2 buckets</td>
</tr>
<tr>
<td>bl256</td>
<td>List with 256 buckets</td>
</tr>
<tr>
<td>bl32</td>
<td>List with 32 buckets</td>
</tr>
<tr>
<td>bl4</td>
<td>List with 4 buckets</td>
</tr>
<tr>
<td>bl64</td>
<td>List with 64 buckets</td>
</tr>
<tr>
<td>bl8</td>
<td>List with 8 buckets</td>
</tr>
</tbody>
</table>
TBucketListSizes is used to set the bucket list size: It specified the number of buckets created by TBucketList (206).

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

TBucketProc is the prototype for the TCustomBucketList.Foreach (215) 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 (203) prototype.

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

TBucketProcObject is the prototype for the TCustomBucketList.Foreach (215) 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 (203) prototype.

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

TDataIteratorCallBack is a callback prototype for the TFPDataHashTable.Iterate (221) 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 (221) 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.

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 (202) class.
THTCustomNodeClass = Class of THTCustomNode

THTCustomNodeClass was used by TFPCustomHashTable (215) 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 (221) 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 (247) link call when 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)

TObjectListStaticCallback is used as the prototype for the TFPObjectList.ForEachCall (247) 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 (249) 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 (249) method. It is called for each element in the hash table, with the string. If Continue is set to False, the iteration stops.

### 8.4 Procedures and functions

#### 8.4.1 RSHash

**Synopsis:** Standard hash value calculating function.

**Declaration:**

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

**Visibility:** default

**Description:**

RSHash is the standard hash calculating function used in the TFPCustomHashTable (215) hash class. It's Robert Sedgwick's "Algorithms in C" hash function.

**Errors:** None.

**See also:** TFPCustomHashTable (215)

#### 8.5 TBucket

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

TBucket describes 1 bucket in the TCustomBucketList (212) class. It is a container for TBucketItem (205) records. It should never be used directly.

#### 8.6 TBucketItem

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

TBucketItem is a record used for internal use in TCustomBucketList (212). It should not be necessary to use it directly.
8.7 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.

8.8 EDuplicate

8.8.1 Description

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

8.9 EKeyNotFound

8.9.1 Description

Exception raised when a key is not found.

See also: TFPCustomHashTable.Delete (218)

8.10 TBucketList

8.10.1 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 (212)

8.10.2 Method overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>206</td>
<td>Create</td>
<td>Create a new TBucketList instance.</td>
</tr>
</tbody>
</table>

8.10.3 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 (202)

### 8.11 TClassList

#### 8.11.1 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 (209), TObjectList (254)

#### 8.11.2 Method overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>207</td>
<td>Add</td>
<td>Add a new class pointer to the list.</td>
</tr>
<tr>
<td>208</td>
<td>Extract</td>
<td>Extract a class pointer from the list.</td>
</tr>
<tr>
<td>208</td>
<td>First</td>
<td>Returns the first non-nil class pointer</td>
</tr>
<tr>
<td>208</td>
<td>IndexOf</td>
<td>Search for a class pointer in the list.</td>
</tr>
<tr>
<td>209</td>
<td>Insert</td>
<td>Insert a new class pointer in the list.</td>
</tr>
<tr>
<td>209</td>
<td>Last</td>
<td>Return last non-nil class pointer</td>
</tr>
<tr>
<td>208</td>
<td>Remove</td>
<td>Remove a class pointer from the list.</td>
</tr>
</tbody>
</table>

#### 8.11.3 Property overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Properties</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>209</td>
<td>Items</td>
<td>rw</td>
<td>Index based access to class pointers.</td>
</tr>
</tbody>
</table>

#### 8.11.4 TClassList.Add

**Synopsis:** Add a new class pointer to the list.

**Declaration:**

```pascal
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 (??) behaviour, 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 (208), #rtl.classes.tlist.add (??)
8.11.5 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 (208), `rtl.classes.Tlist.Extract` (??)

8.11.6 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 (208), `rtl.classes.Tlist.Remove` (??)

8.11.7 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 its position if it was found, or `-1` if it was not found in the list.

Errors: None.

See also: `rtl.classes.tlist.indexof` (??)

8.11.8 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 (209)
8.11.9  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 (208)

8.11.10 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 (207), TClassList.Remove (208)

8.11.11 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 (?)

8.12  TComponentList

8.12.1 Description

TComponentList is a TObjectList (254) 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 (254), 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 (241), TObjectList (254), TClassList (207)
8.12.2 Method overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>210</td>
<td>Add</td>
<td>Add a component to the list.</td>
</tr>
<tr>
<td>210</td>
<td>Destroy</td>
<td>Destroys the instance</td>
</tr>
<tr>
<td>210</td>
<td>Extract</td>
<td>Remove a component from the list without destroying it.</td>
</tr>
<tr>
<td>211</td>
<td>First</td>
<td>First non-nil instance in the list.</td>
</tr>
<tr>
<td>211</td>
<td>IndexOf</td>
<td>Search for an instance in the list</td>
</tr>
<tr>
<td>212</td>
<td>Insert</td>
<td>Insert a new component in the list</td>
</tr>
<tr>
<td>212</td>
<td>Last</td>
<td>Last non-nil instance in the list</td>
</tr>
<tr>
<td>211</td>
<td>Remove</td>
<td>Remove a component from the list, possibly destroying it.</td>
</tr>
</tbody>
</table>

8.12.3 Property overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Properties</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>212</td>
<td>Items</td>
<td>rw</td>
<td>Index-based access to the elements in the list.</td>
</tr>
</tbody>
</table>

8.12.4 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 (254), #rtl.classes.TComponent (??)

8.12.5 TComponentList.Add

Synopsis: Add a component to the list.

Declaration: function Add(AComponent: TComponent) : Integer

Visibility: public

Description: Add overrides the Add operation of its 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 (255)

8.12.6 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 (254) so only TComponent descendants 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 (211), TObjectList.Extract (255)

### 8.12.7 TComponentList.Remove

**Synopsis:** Remove a component from the list, possibly destroying it.

**Declaration:**

```pascal
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 (254) so it only accepts TComponent descendants. It introduces no new behaviour.

**Errors:** None.

See also: TComponentList.Extract (210), TObjectList.Remove (256)

### 8.12.8 TComponentList.IndexOf

**Synopsis:** Search for an instance in the list

**Declaration:**

```pascal
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 (256)

### 8.12.9 TComponentList.First

**Synopsis:** First non-nil instance in the list.

**Declaration:**

```pascal
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 (212), TObjectList.First (257)
8.12.10 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 (211), TObjectList.Last (257)

8.12.11 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 (257), TComponentList.Add (210), TComponentList.Remove (211)

8.12.12 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 (258)

8.13 TCustomBucketList

8.13.1 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 (206). 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 (212).

See also: TBucketList (206)

8.13.2 Method overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>214</td>
<td>Add</td>
<td>Add an item to the list</td>
</tr>
<tr>
<td>214</td>
<td>Assign</td>
<td>Assign one bucket list to another</td>
</tr>
<tr>
<td>213</td>
<td>Clear</td>
<td>Clear the list</td>
</tr>
<tr>
<td>213</td>
<td>Destroy</td>
<td>Frees the bucketlist from memory</td>
</tr>
<tr>
<td>214</td>
<td>Exists</td>
<td>Check if an item exists in the list.</td>
</tr>
<tr>
<td>214</td>
<td>Find</td>
<td>Find an item in the list</td>
</tr>
<tr>
<td>215</td>
<td>ForEach</td>
<td>Loop over all items.</td>
</tr>
<tr>
<td>215</td>
<td>Remove</td>
<td>Remove an item from the list.</td>
</tr>
</tbody>
</table>

8.13.3 Property overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Properties</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>215</td>
<td>Data</td>
<td>rw</td>
<td>Associative array for data pointers</td>
</tr>
</tbody>
</table>

8.13.4 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.

8.13.5 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 (214)
8.13.6 TCustomBucketList.Add

Synopsis: Add an item to the list

Declaration: function Add(AItem: Pointer; AData: Pointer) : Pointer

Visibility: public

Description: Add adds AItem with its 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 (214), TCustomBucketList.Clear (213)

8.13.7 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 (214), TCustomBucketList.Clear (213)

8.13.8 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 (214).

See also: TCustomBucketList.Find (214)

8.13.9 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 (214)
8.13.10 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 (215)

8.13.11 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 (214)

8.13.12 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 (214), TCustomBucketList.Exists (214)

8.14 TFPCustomHashTable

8.14.1 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 (205).

The TFPHashList (222) 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 (250), TFObjectList (241), RSHash (205)

### 8.14.2 Method overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>217</td>
<td>ChangeTableSize</td>
<td>Change the table size of the hash table.</td>
</tr>
<tr>
<td>217</td>
<td>Clear</td>
<td>Clear the hash table.</td>
</tr>
<tr>
<td>216</td>
<td>Create</td>
<td>Instantiate a new TFPCustomHashTable instance using the default hash mechanism</td>
</tr>
<tr>
<td>217</td>
<td>CreateWith</td>
<td>Instantiate a new TFPCustomHashTable instance with given algorithm and size</td>
</tr>
<tr>
<td>218</td>
<td>Delete</td>
<td>Delete a key from the hash list.</td>
</tr>
<tr>
<td>217</td>
<td>Destroy</td>
<td>Free the hash table.</td>
</tr>
<tr>
<td>218</td>
<td>Find</td>
<td>Search for an item with a certain key value.</td>
</tr>
<tr>
<td>218</td>
<td>IsEmpty</td>
<td>Check if the hash table is empty.</td>
</tr>
</tbody>
</table>

### 8.14.3 Property overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Properties</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>220</td>
<td>AVGChainLen</td>
<td>r</td>
<td>Average chain length</td>
</tr>
<tr>
<td>219</td>
<td>Count</td>
<td>r</td>
<td>Number of items in the hash table.</td>
</tr>
<tr>
<td>221</td>
<td>Density</td>
<td>r</td>
<td>Number of filled slots</td>
</tr>
<tr>
<td>218</td>
<td>HashFunction</td>
<td>rw</td>
<td>Hash function currently in use</td>
</tr>
<tr>
<td>219</td>
<td>HashTable</td>
<td>r</td>
<td>Hash table instance</td>
</tr>
<tr>
<td>219</td>
<td>HashTableSize</td>
<td>rw</td>
<td>Size of the hash table</td>
</tr>
<tr>
<td>220</td>
<td>LoadFactor</td>
<td>r</td>
<td>Fraction of count versus size</td>
</tr>
<tr>
<td>220</td>
<td>MaxChainLength</td>
<td>r</td>
<td>Maximum chain length</td>
</tr>
<tr>
<td>220</td>
<td>NumberOfCollisions</td>
<td>r</td>
<td>Number of extra items</td>
</tr>
<tr>
<td>219</td>
<td>VoidSlots</td>
<td>r</td>
<td>Number of empty slots in the hash table.</td>
</tr>
</tbody>
</table>

### 8.14.4 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 (205)

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

See also: CreateWith (217)
8.14.5 TFPHashTable.CreateWith

Synopsis: Instantiate a new TFPHashTable instance with given algorithm and size

Declaration: constructor CreateWith(AHashTableSize: LongWord; aHashFunc: THashFunction)

Visibility: public

Description: CreateWith creates a new instance of TFPHashTable with hash size AHashTableSize and hash calculating algorithm aHashFunc.

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

See also: Create (216)

8.14.6 TFPHashTable.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 (217), Create (216), CreateWith (217)

8.14.7 TFPHashTable.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 (219)

8.14.8 TFPHashTable.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 (217)
8.14.9  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 (218)

8.14.10  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 (250) 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 (218) method should be used instead.

Errors: None.

See also: Delete (218)

8.14.11  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 (219), TFPCustomHashTable.HashTableSize (219), TFPCustomHashTable.AVGChainLen (220), TFPCustomHashTable.MaxChainLength (220)

8.14.12  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 (219)
8.14.13 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 (218), TFPCustomHashTable.HashTableSize (219), TFPCustomHashTable.AVGChainLen (220), TFPCustomHashTable.MaxChainLength (220)

8.14.14 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 (218), TFPCustomHashTable.Count (219), TFPCustomHashTable.AVGChainLen (220), TFPCustomHashTable.MaxChainLength (220), TFPCustomHashTable.VoidSlots (219), TFPCustomHashTable.Density (221)

8.14.15 TFPCustomHashTable.HashTable

Synopsis: Hash table instance

Declaration: Property HashTable : TFPObjectList
- Visibility: public
- Access: Read

Description: TFPCustomHashTable is the internal list object (TFPObjectList (241) used for the hash table. Each element in this table is again a TFPObjectList (241) instance or Nil.

8.14.16 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 (218), TFPCustomHashTable.Count (219), TFPCustomHashTable.AVGChainLen (220), TFPCustomHashTable.MaxChainLength (220), TFPCustomHashTable.LoadFactor (220), TFPCustomHashTable.Density (221), TFPCustomHashTable.NumberOfCollisions (220)
8.14.17  **TFPCustomHashTable.LoadFactor**

**Synopsis:** Fraction of count versus size

**Declaration:**

```pascal
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 (218), TFPCustomHashTable.Count (219), TFPCustomHashTable.AVGChainLen (220), TFPCustomHashTable.MaxChainLength (220), TFPCustomHashTable.VoidSlots (219), TFPCustomHashTable.Density (221), TFPCustomHashTable.NumberOfCollisions (220)

8.14.18  **TFPCustomHashTable.AVGChainLen**

**Synopsis:** Average chain length

**Declaration:**

```pascal
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 (218), TFPCustomHashTable.Count (219), TFPCustomHashTable.LoadFactor (220), TFPCustomHashTable.MaxChainLength (220), TFPCustomHashTable.VoidSlots (219), TFPCustomHashTable.Density (221), TFPCustomHashTable.NumberOfCollisions (220)

8.14.19  **TFPCustomHashTable.MaxChainLength**

**Synopsis:** Maximum chain length

**Declaration:**

```pascal
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 (218), TFPCustomHashTable.Count (219), TFPCustomHashTable.LoadFactor (220), TFPCustomHashTable.AVGChainLen (220), TFPCustomHashTable.VoidSlots (219), TFPCustomHashTable.Density (221), TFPCustomHashTable.NumberOfCollisions (220)

8.14.20  **TFPCustomHashTable.NumberOfCollisions**

**Synopsis:** Number of extra items

**Declaration:**

```pascal
Property NumberOfCollisions : LongWord
```

**Visibility:** public
CHAPTER 8. REFERENCE FOR UNIT „CONTNRS“

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 (218), TFPCustomHashTable.Count (219), TFPCustomHashTable.LoadFactor (220), TFPCustomHashTable.AVGChainLen (220), TFPCustomHashTable.VoidSlots (219), TFPCustomHashTable.Density (221)

8.14.21 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 (218), TFPCustomHashTable.Count (219), TFPCustomHashTable.LoadFactor (220), TFPCustomHashTable.AVGChainLen (220), TFPCustomHashTable.VoidSlots (219), TFPCustomHashTable.Density (221)

8.15 TFPDataHashTable

8.15.1 Description

TFPDataHashTable is a TFPCustomHashTable (215) 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 (239), or for string data, TFPStringHashTable (248) is more suitable. The data pointers are exposed with their keys through the Items (222) property.

See also: TFPObjectHashTable (239), TFPStringHashTable (248), Items (222)

8.15.2 Method overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>222</td>
<td>Add</td>
<td>Add a data pointer to the list.</td>
</tr>
<tr>
<td>221</td>
<td>Iterate</td>
<td>Iterate over the pointers in the hash table</td>
</tr>
</tbody>
</table>

8.15.3 Property overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Properties</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>222</td>
<td>Items</td>
<td>rw</td>
<td>Key-based access to the items in the table</td>
</tr>
</tbody>
</table>

8.15.4 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.

8.15.5 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 (222)

8.15.6 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 a `Nil` pointer.

See also: TFPStringHashTable.Add (249)

8.16 TFPHashList

8.16.1 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 (215) class can be used. If a hash class for objects is needed instead of pointers, the TFPHashObjectList (232) class can be used.

See also: TFPCustomHashTable (215), TFPHashObjectList (232), TFPDataHashTable (221), TFPStringHashTable (248)
8.16.2 Method overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>224</td>
<td>Add</td>
<td>Add a new key/data pair to the list</td>
</tr>
<tr>
<td>224</td>
<td>Clear</td>
<td>Clear the list</td>
</tr>
<tr>
<td>223</td>
<td>Create</td>
<td>Create a new instance of the hashlist</td>
</tr>
<tr>
<td>225</td>
<td>Delete</td>
<td>Delete an item from the list.</td>
</tr>
<tr>
<td>223</td>
<td>Destroy</td>
<td>Removes an instance of the hashlist from the heap</td>
</tr>
<tr>
<td>225</td>
<td>Error</td>
<td>Raise an error</td>
</tr>
<tr>
<td>225</td>
<td>Expand</td>
<td>Expand the list</td>
</tr>
<tr>
<td>225</td>
<td>Extract</td>
<td>Extract a pointer from the list</td>
</tr>
<tr>
<td>226</td>
<td>Find</td>
<td>Find data associated with key</td>
</tr>
<tr>
<td>226</td>
<td>FindIndexOf</td>
<td>Return index of named item.</td>
</tr>
<tr>
<td>226</td>
<td>FindWithHash</td>
<td>Find first element with given name and hash value</td>
</tr>
<tr>
<td>228</td>
<td>ForEachCall</td>
<td>Call a procedure for each element in the list</td>
</tr>
<tr>
<td>225</td>
<td>GetNextCollision</td>
<td>Get next collision number</td>
</tr>
<tr>
<td>224</td>
<td>HashOfIndex</td>
<td>Return the hash value of an item by index</td>
</tr>
<tr>
<td>226</td>
<td>IndexOf</td>
<td>Return the index of the data pointer</td>
</tr>
<tr>
<td>224</td>
<td>NameOfIndex</td>
<td>Returns the key name of an item by index</td>
</tr>
<tr>
<td>227</td>
<td>Pack</td>
<td>Remove nil pointers from the list</td>
</tr>
<tr>
<td>227</td>
<td>Remove</td>
<td>Remove first instance of a pointer</td>
</tr>
<tr>
<td>227</td>
<td>Rename</td>
<td>Rename a key</td>
</tr>
<tr>
<td>227</td>
<td>ShowStatistics</td>
<td>Return some statistics for the list.</td>
</tr>
</tbody>
</table>

8.16.3 Property overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Properties</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>228</td>
<td>Capacity</td>
<td>rw</td>
<td>Capacity of the list.</td>
</tr>
<tr>
<td>228</td>
<td>Count</td>
<td>rw</td>
<td>Current number of elements in the list.</td>
</tr>
<tr>
<td>228</td>
<td>Items</td>
<td>rw</td>
<td>Indexed array with pointers</td>
</tr>
<tr>
<td>229</td>
<td>List</td>
<td>r</td>
<td>Low-level hash list</td>
</tr>
<tr>
<td>229</td>
<td>Strs</td>
<td>r</td>
<td>Low-level memory area with strings.</td>
</tr>
</tbody>
</table>

8.16.4 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 (223)

8.16.5 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.
8.16.6 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 (225), TFPHashList.Remove (227), TFPHashList.Delete (225)

8.16.7 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 (225), TFPHashList.Remove (227), TFPHashList.Delete (225), TFPHashList.Add (224)

8.16.8 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 (224), TFPHashList.Find (226), TFPHashList.FindIndexOf (226), TFPHashList.FindWithHash (226)

8.16.9 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 (224), TFPHashList.Find (226), TFPHashList.FindIndexOf (226), TFPHashList.FindWithHash (226)
8.16.10  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

8.16.11  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 (225), TFPHashList.Remove (227), TFPHashList.Add (224)

8.16.12  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.

8.16.13  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 (224)

8.16.14  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 (225), TFPHashList.Remove (227), TFPHashList.Clear (224)
8.16.15 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 (224), TFPHashList.NameOfIndex (224), TFPHashList.Find (226), TFPHashList.FindIndexOf (226), TFPHashList.FindWithHash (226)

8.16.16 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 (224), TFPHashList.NameOfIndex (224), TFPHashList.IndexOf (226), TFPHashList.FindIndexOf (226), TFPHashList.FindWithHash (226)

8.16.17 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 (224), TFPHashList.NameOfIndex (224), TFPHashList.IndexOf (226), TFPHashList.FindIndexOf (226), TFPHashList.FindWithHash (226)

8.16.18 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 (224), TFPHashList.NameOfIndex (224), TFPHashList.IndexOf (226), TFPHashList.Find (226), TFPHashList.FindIndexOf (226)
8.16.19 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.

8.16.20 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 (225), TFPHashList.Clear (224), TFPHashList.Extract (225)

8.16.21 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 (224)

8.16.22 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:

- **HashSize**: Size of the hash table
- **HashMean**: Mean hash value
- **HashStdDev**: Standard deviation of hash values
- **ListSize**: Size and capacity of the list
- **StringSize**: Size and capacity of key strings
8.16.23 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.

8.16.24 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 (228), Items (228)

8.16.25 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 (228), Items (228)

8.16.26 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 (228).

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

See also: Capacity (228), Count (228)
8.16.27 TFPHashList.List

Synopsis: Low-level hash list

Declaration: Property List : PHashItemList

Visibility: public

Access: Read

Description: List exposes the low-level item list (203). It should not be used directly.

See also: Strs (229), THashItemList (203)

8.16.28 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 (229)

8.17 TFPHashObject

8.17.1 Description

TFPHashObject is a TObject descendent which is aware of the TFPHashObjectList (232) 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 (231), TFPHashObject.ChangeOwner (230), TFPHashObject.ChangeOwnerAndName (230)

8.17.2 Method overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>230</td>
<td>ChangeOwner</td>
<td>Change the list owning the object.</td>
</tr>
<tr>
<td>230</td>
<td>ChangeOwnerAndName</td>
<td>Simultaneously change the list owning the object and the name of the object.</td>
</tr>
<tr>
<td>230</td>
<td>Create</td>
<td>Create a named instance, and insert in a hash list.</td>
</tr>
<tr>
<td>230</td>
<td>CreateNotOwned</td>
<td>Create an instance not owned by any list.</td>
</tr>
<tr>
<td>231</td>
<td>Rename</td>
<td>Rename the object</td>
</tr>
</tbody>
</table>

8.17.3 Property overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Properties</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>231</td>
<td>Hash</td>
<td>r</td>
<td>Hash value</td>
</tr>
<tr>
<td>231</td>
<td>Name</td>
<td>r</td>
<td>Current name of the object</td>
</tr>
</tbody>
</table>
8.17.4 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 (232) hash list. It also has no name when created in this way.

See also: TFPHashObject.Name (231), TFPHashObject.ChangeOwner (230), TFPHashObject.ChangeOwnerAndName (230)

8.17.5 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 (232).

See also: CreateNotOwned (230), TFPHashObject.ChangeOwner (230), TFPHashObject.Name (231)

8.17.6 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 (230), Name (231)

8.17.7 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 (230), Name (231)
8.17.8 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 (230), ChangeOwnerAndName (230), Name (231)

8.17.9 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 (231), ChangeOwnerAndName (230)

8.17.10 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 (231)
8.18 TFPHashObjectList

8.18.1 Method overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>233</td>
<td>Add</td>
<td>Add a new key/data pair to the list</td>
</tr>
<tr>
<td>233</td>
<td>Clear</td>
<td>Clear the list</td>
</tr>
<tr>
<td>232</td>
<td>Create</td>
<td>Create a new instance of the hashlist</td>
</tr>
<tr>
<td>234</td>
<td>Delete</td>
<td>Delete an object from the list</td>
</tr>
<tr>
<td>232</td>
<td>Destroy</td>
<td>Removes an instance of the hashlist from the heap</td>
</tr>
<tr>
<td>234</td>
<td>Expand</td>
<td>Expand the list</td>
</tr>
<tr>
<td>235</td>
<td>Extract</td>
<td>Extract a object instance from the list</td>
</tr>
<tr>
<td>235</td>
<td>Find</td>
<td>Find data associated with key</td>
</tr>
<tr>
<td>236</td>
<td>FindIndexOf</td>
<td>Return index of named object.</td>
</tr>
<tr>
<td>236</td>
<td>FindInstanceOf</td>
<td>Search an instance of a certain class</td>
</tr>
<tr>
<td>236</td>
<td>FindWithHash</td>
<td>Find first element with given name and hash value</td>
</tr>
<tr>
<td>237</td>
<td>ForEachCall</td>
<td>Call a procedure for each object in the list</td>
</tr>
<tr>
<td>234</td>
<td>GetNextCollision</td>
<td>Get next collision number</td>
</tr>
<tr>
<td>234</td>
<td>HashOfIndex</td>
<td>Return the hash value of an object by index</td>
</tr>
<tr>
<td>235</td>
<td>IndexOf</td>
<td>Return the index of the object instance</td>
</tr>
<tr>
<td>233</td>
<td>NameOfIndex</td>
<td>Returns the key name of an object by index</td>
</tr>
<tr>
<td>237</td>
<td>Pack</td>
<td>Remove nil object instances from the list</td>
</tr>
<tr>
<td>235</td>
<td>Remove</td>
<td>Remove first occurrence of a object instance</td>
</tr>
<tr>
<td>236</td>
<td>Rename</td>
<td>Rename a key</td>
</tr>
<tr>
<td>237</td>
<td>ShowStatistics</td>
<td>Return some statistics for the list.</td>
</tr>
</tbody>
</table>

8.18.2 Property overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Properties</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>237</td>
<td>Capacity</td>
<td>rw</td>
<td>Capacity of the list.</td>
</tr>
<tr>
<td>238</td>
<td>Count</td>
<td>rw</td>
<td>Current number of elements in the list.</td>
</tr>
<tr>
<td>238</td>
<td>Items</td>
<td>rw</td>
<td>Indexed array with object instances</td>
</tr>
<tr>
<td>238</td>
<td>List</td>
<td>r</td>
<td>Low-level hash list</td>
</tr>
<tr>
<td>238</td>
<td>OwnsObjects</td>
<td>rw</td>
<td>Does the list own the objects it contains</td>
</tr>
</tbody>
</table>

8.18.3 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 (232), TFPHashObjectList.OwnsObjects (238)

8.18.4 TFPHashObjectList.Destroy

Synopsis: Removes an instance of the hashlist from the heap
CHAPTER 8. REFERENCE FOR UNIT 'CONTNRS'

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 (232), TFPHashObjectList.Clear (233)

8.18.5 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 (238) is True. It always frees all memory needed to contain the objects.

Errors: None.

See also: TFPHashObjectList.Extract (235), TFPHashObjectList.Remove (235), TFPHashObjectList.Delete (234), TFPHashObjectList.Add (233)

8.18.6 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 (235), TFPHashObjectList.Remove (235), TFPHashObjectList.Delete (234)

8.18.7 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 (234), TFPHashObjectList.Find (235), TFPHashObjectList.FindIndexOf (236), TFPHashObjectList.FindWithHash (236)
8.18.8 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 (233), TFPHashObjectList.Find (235), TFPHashObjectList.FindIndexOf (236), TFPHashObjectList.FindWithHash (236)

8.18.9 TFPHashObjectList.GetNextCollision
Synopsis: Get next collision number

Declaration: function GetNextCollision(Index: Integer) : Integer
Visibility: public

Description: Get next collision number

8.18.10 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 (238) is True, then the object itself is also freed from memory.

See also: TFPHashObjectList.Extract (235), TFPHashObjectList.Remove (235), TFPHashObjectList.Add (233), OwnsObjects (238)

8.18.11 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 (233)
8.18.12 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 (238).

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

See also: TFPHashObjectList.Delete (234), TFPHashObjectList.Remove (235), TFPHashObjectList.Clear (233)

8.18.13 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 (238) is True, then the object itself is also freed from memory.

See also: TFPHashObjectList.Delete (234), TFPHashObjectList.Clear (233), TFPHashObjectList.Extract (235)

8.18.14 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 (234), TFPHashObjectList.NameOfIndex (233), TFPHashObjectList.Find (235), TFPHashObjectList.FindIndexOf (236), TFPHashObjectList.FindWithHash (236)

8.18.15 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 (234), TFPHashObjectList.NameOfIndex (233), TFPHashObjectList.IndexOf (235), TFPHashObjectList.FindIndexOf (236), TFPHashObjectList.FindWithHash (236)
8.18.16 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 (234), TFPHashObjectList.NameOfIndex (233), TFPHashObjectList.IndexOf (235), TFPHashObjectList.Find (235), TFPHashObjectList.FindWithHash (236)

8.18.17 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 (234), TFPHashObjectList.NameOfIndex (233), TFPHashObjectList.IndexOf (235), TFPHashObjectList.Find (235), TFPHashObjectList.FindIndexOf (236)

8.18.18 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 its new position.

Errors: If an object with ANewName already exists, an exception will be raised.

8.18.19 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.
8.18.20  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 (233)

8.18.21  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:

- HashSize: Size of the hash table
- HashMean: Mean hash value
- HashStdDev: Standard deviation of hash values
- ListSize: Size and capacity of the list
- StringSize: Size and capacity of key strings

8.18.22  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.

8.18.23  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 (238), Items (238)
8.18.24  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 (237), Items (238)

8.18.25  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 (232)

8.18.26  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 (238).
   Errors: Specifying an invalid index will result in an exception.
See also: Capacity (237), Count (238)

8.18.27  TFPHashObjectList.List
Synopsis: Low-level hash list
Declaration: Property List : TFPHashList
   Visibility: public
   Access: Read
Description: List exposes the low-level hash list (222). It should not be used directly.
See also: TFPHashList (222)
8.19  TFPObjectHashTable

8.19.1  Description
TFPStringHashTable is a TFPCustomHashTable (215) 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 (248), or for arbitrary pointer data, TFPDataHashTable (221) is more suitable. The objects are exposed with their keys through the Items (240) property.

See also: TFPStringHashTable (248), TFPDataHashTable (221), TFPObjectHashTable.Items (240)

8.19.2  Method overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>240</td>
<td>Add</td>
<td>Add a new object to the hash table</td>
</tr>
<tr>
<td>239</td>
<td>Create</td>
<td>Create a new instance of TFPObjectHashTable</td>
</tr>
<tr>
<td>239</td>
<td>CreateWith</td>
<td>Create a new hash table with given size and hash function</td>
</tr>
<tr>
<td>240</td>
<td>Iterate</td>
<td>Iterate over the objects in the hash table</td>
</tr>
</tbody>
</table>

8.19.3  Property overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Properties</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>240</td>
<td>Items</td>
<td>rw</td>
<td>Key-based access to the objects</td>
</tr>
<tr>
<td>240</td>
<td>OwnsObjects</td>
<td>r</td>
<td>Does the hash table own the objects?</td>
</tr>
</tbody>
</table>

8.19.4  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 (240) 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 (240), TFPObjectHashTable.CreateWith (239), TFPObjectHashTable.Items (240)

8.19.5  TFPObjectHashTable.CreateWith

Synopsis: Create a new hash table with given size and hash function

Declaration: constructor CreateWith(AHashTableSize: LongWord; aHashFunc: THashTable; AOwnsObjects: Boolean)

Visibility: public

Description: CreateWith sets the OwnsObjects (240) 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 (240), TFPObjectHashTable.Create (239), TFPObjectHashTable.Items
(240)

8.19.6 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.

8.19.7 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 (240)

8.19.8 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 (240)

8.19.9 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 (239), TFPObjectHashTable.Items (240)

8.20 TFPObjectList

8.20.1 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 (254), 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 (254)

8.20.2 Method overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>242</td>
<td>Add</td>
<td>Add an object to the list.</td>
</tr>
<tr>
<td>246</td>
<td>Assign</td>
<td>Copy the contents of a list.</td>
</tr>
<tr>
<td>242</td>
<td>Clear</td>
<td>Clear all elements in the list.</td>
</tr>
<tr>
<td>242</td>
<td>Create</td>
<td>Create a new object list</td>
</tr>
<tr>
<td>243</td>
<td>Delete</td>
<td>Delete an element from the list.</td>
</tr>
<tr>
<td>242</td>
<td>Destroy</td>
<td>Clears the list and destroys the list instance</td>
</tr>
<tr>
<td>243</td>
<td>Exchange</td>
<td>Exchange the location of two objects</td>
</tr>
<tr>
<td>243</td>
<td>Expand</td>
<td>Expand the capacity of the list.</td>
</tr>
<tr>
<td>244</td>
<td>Extract</td>
<td>Extract an object from the list</td>
</tr>
<tr>
<td>244</td>
<td>FindInstanceOf</td>
<td>Search for an instance of a certain class</td>
</tr>
<tr>
<td>245</td>
<td>First</td>
<td>Return the first non-nil object in the list</td>
</tr>
<tr>
<td>247</td>
<td>ForEachCall</td>
<td>For each object in the list, call a method or procedure, passing it the object.</td>
</tr>
<tr>
<td>244</td>
<td>IndexOf</td>
<td>Search for an object in the list</td>
</tr>
<tr>
<td>245</td>
<td>Insert</td>
<td>Insert a new object in the list</td>
</tr>
<tr>
<td>245</td>
<td>Last</td>
<td>Return the last non-nil object in the list</td>
</tr>
<tr>
<td>246</td>
<td>Move</td>
<td>Move an object to another location in the list.</td>
</tr>
<tr>
<td>246</td>
<td>Pack</td>
<td>Remove all Nil references from the list</td>
</tr>
<tr>
<td>244</td>
<td>Remove</td>
<td>Remove an item from the list.</td>
</tr>
<tr>
<td>246</td>
<td>Sort</td>
<td>Sort the list of objects</td>
</tr>
</tbody>
</table>
8.20.3 Property overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Properties</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>247</td>
<td>Capacity</td>
<td>rw</td>
<td>Capacity of the list</td>
</tr>
<tr>
<td>247</td>
<td>Count</td>
<td>rw</td>
<td>Number of elements in the list.</td>
</tr>
<tr>
<td>248</td>
<td>Items</td>
<td>rw</td>
<td>Indexed access to the elements of the list.</td>
</tr>
<tr>
<td>248</td>
<td>List</td>
<td>r</td>
<td>Internal list used to keep the objects.</td>
</tr>
<tr>
<td>248</td>
<td>OwnsObjects</td>
<td>rw</td>
<td>Should the list free elements when they are removed.</td>
</tr>
</tbody>
</table>

8.20.4 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.

See also: TFPObjectList.Destroy (242), TFPObjectList.OwnsObjects (248), TObjectList (254)

8.20.5 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 (248) is True.

See also: TFPObjectList.OwnsObjects (248), TObjectList.Create (255)

8.20.6 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 (248) is True.

8.20.7 TFPObjectList.Add

Synopsis: Add an object to the list.

Declaration: function Add(AObject: TObject) : Integer

Visibility: public
CHAPTER 8. REFERENCE FOR UNIT 'CONTNRS'

Description: **Add** adds an object to the list and returns the index of the object in the list.

Note that when **OwnsObjects** (248) 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** (248), TFPObjectList.**Delete** (243)

### 8.20.8 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** (248) is True, the object is also freed.

Errors: An access violation may occur when **OwnsObjects** (248) is True and either the object was freed externally, or when the same object is in the same list twice.

See also: TFPObjectList.**Remove** (244), TFPObjectList.**Extract** (244), TFPObjectList.**OwnsObjects** (248), TFPObjectList.**Add** (242), TFPObjectList.**Clear** (242)

### 8.20.9 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** (242), TFPObjectList.**Delete** (243)

### 8.20.10 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** (246), TFPObjectList.**Clear** (242), **rtl.classes.tfplist.expand** (??)
8.20.11 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 (246), TFPObjectList.Clear (242), TFPObjectList.Remove (244), TFPObjectList.Delete (243)

8.20.12 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 (248) 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 (246), TFPObjectList.Clear (242), TFPObjectList.Delete (243), TFPObjectList.Extract (244)

8.20.13 TFPObjectList.IndexOf

Synopsis: Search for an object in the list

Declaration: functionIndexOf(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 (248), TFPObjectList.Remove (244), TFPObjectList.Extract (244)

8.20.14 TFPObjectList.FindInstanceOf

Synopsis: Search for an instance of a certain class

Declaration: function FindInstanceOf(AClass: TClass; AExact: Boolean; AStartAt: Integer) : Integer

Visibility: public
**CHAPTER 8. REFERENCE FOR UNIT 'CONTNRS'**

**Description:** \texttt{FindInstanceOf} will look through the instances in the list and will return the first instance which is a descendent of class \texttt{AClass} if \texttt{AExact} is \texttt{False}. If \texttt{AExact} is \texttt{true}, then the instance should be of class \texttt{AClass}.

If no instance of the requested class is found, \texttt{Nil} is returned.

**Errors:** None.

See also: \texttt{TFPObjectList.IndexOf (244)}

---

**8.20.15 TFPObjectList.Insert**

**Synopsis:** Insert a new object in the list

**Declaration:**

\texttt{procedure Insert(Index: Integer; AObject: TObject)}

**Visibility:** public

**Description:** \texttt{Insert} inserts \texttt{AObject} at position \texttt{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: \texttt{TFPObjectList.Add (242)}, \texttt{TFPObjectList.Delete (243)}

---

**8.20.16 TFPObjectList.First**

**Synopsis:** Return the first non-nil object in the list

**Declaration:**

\texttt{function First : TObject}

**Visibility:** public

**Description:** \texttt{First} returns a reference to the first non-\texttt{Nil} element in the list. If no non-\texttt{Nil} element is found, \texttt{Nil} is returned.

**Errors:** None.

See also: \texttt{TFPObjectList.Last (245)}, \texttt{TFPObjectList.Pack (246)}

---

**8.20.17 TFPObjectList.Last**

**Synopsis:** Return the last non-nil object in the list.

**Declaration:**

\texttt{function Last : TObject}

**Visibility:** public

**Description:** \texttt{Last} returns a reference to the last non-\texttt{Nil} element in the list. If no non-\texttt{Nil} element is found, \texttt{Nil} is returned.

**Errors:** None.

See also: \texttt{TFPObjectList.First (245)}, \texttt{TFPObjectList.Pack (246)}
8.20.18 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 (243), 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 (243), TFPObjectList.Delete (243), TFPObjectList.Insert (245)

8.20.19 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.

8.20.20 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 (245), TFPObjectList.Last (245)

8.20.21 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 \texttt{Nil} values.

\textbf{Errors}: None.

\textbf{See also}: \#rtl.classes.TList.Sort (??)

\section*{8.20.22 \texttt{TFPObjectList.ForEachCall}}

\textbf{Synopsis}: For each object in the list, call a method or procedure, passing it the object.

\textbf{Declaration}:
\begin{verbatim}
procedure ForEachCall(proc2call: TObjectListCallback; arg: pointer)
procedure ForEachCall(proc2call: TObjectListStaticCallback;
                      arg: pointer)
\end{verbatim}

\textbf{Visibility}:
public

\textbf{Description}:
\texttt{ForEachCall} loops through all objects in the list, and calls \texttt{proc2call}, passing it the object in the list. Additionally, \texttt{arg} is also passed to the procedure. \texttt{Proc2call} can be a plain procedure or can be a method of a class.

\textbf{Errors}: None.

\textbf{See also}: TObjectListStaticCallback (204), TObjectListCallback (204)

\section*{8.20.23 \texttt{TFPObjectList.Capacity}}

\textbf{Synopsis}: Capacity of the list

\textbf{Declaration}:
\begin{verbatim}
Property Capacity : Integer
\end{verbatim}

\textbf{Visibility}:
public

\textbf{Access}:
Read,Write

\textbf{Description}:
\texttt{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 \texttt{Count} (247).

\textbf{See also}: TFPObjectList.Count (247)

\section*{8.20.24 \texttt{TFPObjectList.Count}}

\textbf{Synopsis}: Number of elements in the list.

\textbf{Declaration}:
\begin{verbatim}
Property Count : Integer
\end{verbatim}

\textbf{Visibility}:
public

\textbf{Access}:
Read,Write

\textbf{Description}:
\texttt{Count} is the number of elements in the list. Note that this includes \texttt{Nil} elements.

\textbf{See also}: TFPObjectList.Capacity (247)
CHAPTER 8. REFERENCE FOR UNIT 'CONTNRS'

8.20.25 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 (242), TFPObjectList.Delete (243), TFPObjectList.Remove (244), TFPObjectList.Clear (242)

8.20.26 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 (247)

8.20.27 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 (??)

8.21 TFPStringHashTable

8.21.1 Description

TFPStringHashTable is a TFPCustomHashTable (215) 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 (239), or for arbitrary pointer data, TFPDataHashTable (221) is more suitable. The strings are exposed with their keys through the Items (249) property.

See also: TFPObjectHashTable (239), TFPDataHashTable (221), Items (249)
8.21.2 Method overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>249</td>
<td>Add</td>
<td>Add a new string to the hash list</td>
</tr>
<tr>
<td>249</td>
<td>Iterate</td>
<td>Iterate over the strings in the hash table</td>
</tr>
</tbody>
</table>

8.21.3 Property overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Properties</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>249</td>
<td>Items</td>
<td>rw</td>
<td>Key based access to the strings in the hash table</td>
</tr>
</tbody>
</table>

8.21.4 TFPStringHashTable.Iterate

**Synopsis:** Iterate over the strings in the hash table

**Declaration:**

```pascal
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.

8.21.5 TFPStringHashTable.Add

**Synopsis:** Add a new string to the hash list

**Declaration:**

```pascal
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 (249)

8.21.6 TFPStringHashTable.Items

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

**Declaration:**

```pascal
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 (249)
8.22  THTCustomNode

8.22.1  Description

THTCustomNode is used by the TFPCustomHashTable (215) class to store the keys and associated values.

See also: TFPCustomHashTable (215)

8.22.2  Method overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>250</td>
<td>CreateWith</td>
<td>Create a new instance of THTCustomNode</td>
</tr>
<tr>
<td>250</td>
<td>HasKey</td>
<td>Check whether this node matches the given key.</td>
</tr>
</tbody>
</table>

8.22.3  Property overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Properties</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>251</td>
<td>Key</td>
<td>r</td>
<td>Key value associated with this hash item.</td>
</tr>
</tbody>
</table>

8.22.4  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 (215) class when needed.

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

See also: TFPCustomHashTable (215)

8.22.5  THTCustomNodeHasKey

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 (251)
8.22.6  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 (250)

8.23  THTDataNode

8.23.1 Description

THTDataNode is used by TFPDataHashTable (221) 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 (221), THTObjectNode (251), THTStringNode (252)

8.23.2 Property overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Properties</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>251</td>
<td>Data</td>
<td>rw</td>
<td>Data pointer</td>
</tr>
</tbody>
</table>

8.23.3 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.

8.24  THTObjectNode

8.24.1 Description

THTObjectNode is a THTCustomNode (250) descendent which holds the data in the TFPObjectHashTable (239) 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 (239)
8.24.2 Property overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Properties</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>252</td>
<td>Data</td>
<td>rw</td>
<td>Object instance</td>
</tr>
</tbody>
</table>

8.24.3 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 (240)  

See also: TFPObjectHashTable (239), TFPObjectHashTable.Items (240), THTOwnedObjectNode (252)

8.25 THTOwnedObjectNode

8.25.1 Description

THTOwnedObjectNode is used instead of THTObjectNode (251) in case TFPObjectHashTable (239) owns its objects. When this object is destroyed, the associated data object is also destroyed.  

See also: TFPObjectHashTable (239), THTObjectNode (251), TFPObjectHashTable.OwnsObjects (240)

8.25.2 Method overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>252</td>
<td>Destroy</td>
<td>Destroys the node and the object.</td>
</tr>
</tbody>
</table>

8.25.3 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 (252), TFPObjectHashTable.OwnsObjects (240)

8.26 THTStringNode

8.26.1 Description

THTStringNode is a THTCustomNode (250) descendent which holds the data in the TFPString-HashTable (248) 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 (248)
8.26.2 Property overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Properties</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>253</td>
<td>Data</td>
<td>rw</td>
<td>String data</td>
</tr>
</tbody>
</table>

8.26.3 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 (249)

See also: TFPStringHashTable (248)

8.27 TObjectBucketList

8.27.1 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 (253)

8.27.2 Method overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>253</td>
<td>Add</td>
<td>Add an object to the list</td>
</tr>
<tr>
<td>254</td>
<td>Remove</td>
<td>Remove an object from the list</td>
</tr>
</tbody>
</table>

8.27.3 Property overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Properties</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>254</td>
<td>Data</td>
<td>rw</td>
<td>Associative array of data items</td>
</tr>
</tbody>
</table>

8.27.4 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 (254), TObjectBucketList.Remove (254)
8.27.5 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 (253), TObjectBucketList.Data (254)

8.27.6 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 (253)

8.28 TObjectList

8.28.1 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 (241), 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 (241), TComponentList (209), TClassList (207)

8.28.2 Method overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>255</td>
<td>Add</td>
<td>Add an object to the list.</td>
</tr>
<tr>
<td>255</td>
<td>Create</td>
<td>Create a new object list.</td>
</tr>
<tr>
<td>255</td>
<td>Extract</td>
<td>Extract an object from the list.</td>
</tr>
<tr>
<td>256</td>
<td>FindInstanceOf</td>
<td>Search for an instance of a certain class</td>
</tr>
<tr>
<td>257</td>
<td>First</td>
<td>Return the first non-nil object in the list</td>
</tr>
<tr>
<td>256</td>
<td>IndexOf</td>
<td>Search for an object in the list</td>
</tr>
<tr>
<td>257</td>
<td>Insert</td>
<td>Insert an object in the list.</td>
</tr>
<tr>
<td>257</td>
<td>Last</td>
<td>Return the last non-nil object in the list</td>
</tr>
<tr>
<td>256</td>
<td>Remove</td>
<td>Remove (and possibly free) an element from the list.</td>
</tr>
</tbody>
</table>
### 8.28.3 Property overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Properties</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>258</td>
<td>Items</td>
<td>rw</td>
<td>Indexed access to the elements of the list.</td>
</tr>
<tr>
<td>257</td>
<td>OwnsObjects</td>
<td>rw</td>
<td>Should the list free elements when they are removed.</td>
</tr>
</tbody>
</table>

### 8.28.4 TObjectList.Create

**Synopsis:** Create a new object list.

**Declaration:**

```plaintext
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 (257), TFPObjectList (241)

### 8.28.5 TObjectList.Add

**Synopsis:** Add an object to the list.

**Declaration:**

```plaintext
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 (257), #rtl.classes.TList.Delete (?), TObjectList.Extract (255), TObjectList.Remove (256)

### 8.28.6 TObjectList.Extract

**Synopsis:** Extract an object from the list.

**Declaration:**

```plaintext
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` (256), `Extract` does not free the extracted element if `OwnsObjects` (257) 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 (256)
8.28.7 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 (257) 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 (255)

8.28.8 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 (256)

8.28.9 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 (256)
8.28.10  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 (255), TObjectList.Remove (256)

8.28.11  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 (257)

8.28.12  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 (257)

8.28.13  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 (255), TObjectList.Remove (256), TObjectList.Extract (255)
8.28.14 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 is zero based, i.e., runs from 0 (zero) to Count-1.

See also: #rtl.classes.TList.Count (??)

8.29 TObjectQueue

8.29.1 Method overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>259</td>
<td>Peek</td>
<td>Look at the first object in the queue.</td>
</tr>
<tr>
<td>258</td>
<td>Pop</td>
<td>Pop the first element off the queue</td>
</tr>
<tr>
<td>258</td>
<td>Push</td>
<td>Push an object on the queue</td>
</tr>
</tbody>
</table>

8.29.2 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 (258), TObjectQueue.Peek (259)

8.29.3 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 (258), TObjectQueue.Peek (259)
### 8.29.4 TObjectQueue.Peek

**Synopsis:** Look at the first object in the queue.

**Declaration:**

```delphi
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 (258), TObjectQueue.Pop (258)

### 8.30 TObjectStack

#### 8.30.1 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 (260), TStack (262), TQueue (262), TObjectQueue (258)

#### 8.30.2 Method overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>260</td>
<td>Peek</td>
<td>Look at the top object in the stack.</td>
</tr>
<tr>
<td>259</td>
<td>Pop</td>
<td>Pop the top object of the stack.</td>
</tr>
<tr>
<td>259</td>
<td>Push</td>
<td>Push an object on the stack.</td>
</tr>
</tbody>
</table>

#### 8.30.3 TObjectStack.Push

**Synopsis:** Push an object on the stack.

**Declaration:**

```delphi
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 (259), TObjectStack.Peek (260)

#### 8.30.4 TObjectStack.Pop

**Synopsis:** Pop the top object of the stack.

**Declaration:**

```delphi
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 (259), TObjectStack.Peek (260)

8.30.5 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 (259), TObjectStack.Pop (259)

8.31 TOrderedList

8.31.1 Description

TOrderedList provides the base class for TQueue (262) and TStack (262). 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 (262), TStack (262)

8.31.2 Method overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>261</td>
<td>AtLeast</td>
<td>Check whether the list contains a certain number of elements.</td>
</tr>
<tr>
<td>261</td>
<td>Count</td>
<td>Number of elements on the list.</td>
</tr>
<tr>
<td>260</td>
<td>Create</td>
<td>Create a new ordered list</td>
</tr>
<tr>
<td>261</td>
<td>Destroy</td>
<td>Free an ordered list</td>
</tr>
<tr>
<td>262</td>
<td>Peek</td>
<td>Return the next element to be popped from the list.</td>
</tr>
<tr>
<td>262</td>
<td>Pop</td>
<td>Remove an element from the list.</td>
</tr>
<tr>
<td>261</td>
<td>Push</td>
<td>Push another element on the list.</td>
</tr>
</tbody>
</table>

8.31.3 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 (261)
8.31.4 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 (260)

8.31.5 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 (261)

8.31.6 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 (261)

8.31.7 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 (262), TOrderedList.Peek (262)
8.31.8 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 (262), TOrderedList.Push (261)

8.31.9 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 (262), without actually popping it from the list.
Errors: None.
See also: TOrderedList.Pop (262), TOrderedList.Push (261)

8.32 TQueue

8.32.1 Description
TQueue is a descendent of TOrderedList (260) which implements Push (261) and Pop (262) 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 (260)
See also: TOrderedList (260), TObjectQueue (258), TStack (262)

8.33 TStack

8.33.1 Description
TStack is a descendent of TOrderedList (260) which implements Push (261) and Pop (262) 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 (260)
See also: TOrderedList (260), TObjectStack (259), TQueue (262)
Chapter 9

Reference for unit ’CustApp’

9.1 Used units

Table 9.1: Used units by unit ’CustApp’

<table>
<thead>
<tr>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Classes</td>
<td>??</td>
</tr>
<tr>
<td>singleinstance</td>
<td>??</td>
</tr>
<tr>
<td>System</td>
<td>??</td>
</tr>
<tr>
<td>sysutils</td>
<td>??</td>
</tr>
</tbody>
</table>

9.2 Overview

The CustApp unit implements the TCustomApplication (264) 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:

```pascal
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.

9.3 Constants, types and variables

9.3.1 Types

`TEventLogTypes = Set of TEventType`
TEventLogTypes is a set of TEventType (?!), used in TCustomApplication.EventLogFilter (275) 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 (269) call.

9.3.2 Variables

CustomApplication : TCustomApplication = Nil

CustomApplication contains the global application instance. All descendents of TCustomApplication (264) 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.

9.4 TCustomApplication

9.4.1 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 (266), ShowException (267), OnException (272) and StopOnException (275).
- Command-line parameter parsing in FindOptionIndex (268), GetOptionValue (268), CheckOptions (269) and HasOption (269)
- Environment variable handling in GetEnvironmentList (271) and EnvironmentVariable (274).

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

<table>
<thead>
<tr>
<th>Page</th>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>269</td>
<td>CheckOptions</td>
<td>Check whether all given options on the command-line are valid.</td>
</tr>
<tr>
<td>265</td>
<td>Create</td>
<td>Create a new instance of the TCustomApplication class</td>
</tr>
<tr>
<td>266</td>
<td>Destroy</td>
<td>Destroys the TCustomApplication instance.</td>
</tr>
<tr>
<td>268</td>
<td>FindOptionIndex</td>
<td>Return the index of an option.</td>
</tr>
<tr>
<td>271</td>
<td>GetEnvironmentList</td>
<td>Return a list of environment variables.</td>
</tr>
<tr>
<td>270</td>
<td>GetNonOptions</td>
<td>Get all non-switch options</td>
</tr>
<tr>
<td>268</td>
<td>GetOptionValue</td>
<td>Return the value of a command-line option.</td>
</tr>
<tr>
<td>269</td>
<td>GetOptionValues</td>
<td>Get the values for an option that may be specified multiple times</td>
</tr>
<tr>
<td>266</td>
<td>HandleException</td>
<td>Handle an exception.</td>
</tr>
<tr>
<td>269</td>
<td>HasOption</td>
<td>Check whether an option was specified.</td>
</tr>
<tr>
<td>266</td>
<td>Initialize</td>
<td>Initialize the application</td>
</tr>
<tr>
<td>271</td>
<td>Log</td>
<td>Write a message to the event log</td>
</tr>
<tr>
<td>267</td>
<td>Run</td>
<td>Runs the application.</td>
</tr>
<tr>
<td>267</td>
<td>ShowException</td>
<td>Show an exception to the user</td>
</tr>
<tr>
<td>267</td>
<td>Terminate</td>
<td>Terminate the application</td>
</tr>
</tbody>
</table>

9.4.3 Property overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Properties</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>274</td>
<td>CaseSensitiveOptions</td>
<td>rw</td>
<td>Are options interpreted case sensitive or not</td>
</tr>
<tr>
<td>273</td>
<td>ConsoleApplication</td>
<td>r</td>
<td>Is the application a console application or not</td>
</tr>
<tr>
<td>274</td>
<td>EnvironmentVariable</td>
<td>r</td>
<td>Environment variable access</td>
</tr>
<tr>
<td>275</td>
<td>EventLogFilter</td>
<td>rw</td>
<td>Event to filter events, before they are sent to the system log</td>
</tr>
<tr>
<td>275</td>
<td>ExceptionExitCode</td>
<td>rw</td>
<td>ExitCode to use then terminating the program due to an exception</td>
</tr>
<tr>
<td>271</td>
<td>ExeName</td>
<td>r</td>
<td>Name of the executable.</td>
</tr>
<tr>
<td>272</td>
<td>HelpFile</td>
<td>rw</td>
<td>Location of the application help file.</td>
</tr>
<tr>
<td>273</td>
<td>Location</td>
<td>r</td>
<td>Application location</td>
</tr>
<tr>
<td>272</td>
<td>OnException</td>
<td>rw</td>
<td>Exception handling event</td>
</tr>
<tr>
<td>274</td>
<td>OptionChar</td>
<td>rw</td>
<td>Command-line switch character</td>
</tr>
<tr>
<td>274</td>
<td>ParamCount</td>
<td>r</td>
<td>Number of command-line parameters</td>
</tr>
<tr>
<td>273</td>
<td>Params</td>
<td>r</td>
<td>Command-line parameters</td>
</tr>
<tr>
<td>276</td>
<td>SingleInstance</td>
<td>r</td>
<td>Single instance used to control single application instance behaviour</td>
</tr>
<tr>
<td>276</td>
<td>SingleInstanceClass</td>
<td>rw</td>
<td>Class to use when creating single instance</td>
</tr>
<tr>
<td>276</td>
<td>SingleInstanceEnabled</td>
<td>rw</td>
<td>Enable single application instance control.</td>
</tr>
<tr>
<td>275</td>
<td>StopOnException</td>
<td>rw</td>
<td>Should the program loop stop on an exception</td>
</tr>
<tr>
<td>272</td>
<td>Terminated</td>
<td>r</td>
<td>Was Terminate called or not</td>
</tr>
<tr>
<td>272</td>
<td>Title</td>
<td>rw</td>
<td>Application title</td>
</tr>
</tbody>
</table>

9.4.4 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.
9.4.5  TCustomApplication.Destroy

Synopsis: Destroys the TCustomApplication instance.

Declaration: destructor Destroy; Override

Visibility: public

Description: Destroy simply calls the inherited Destroy.

See also: TCustomApplication.Create (265)

9.4.6  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 (272) 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 (267)
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 (275) property is set
to True, the Terminated (272) property is set to True, which will cause the Run (267) loop to stop,
and the application will exit.

See also: ShowException (267), StopOnException (275), Terminated (272), Run (267)

9.4.7  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 (267), TCustomApplication.Terminated (272)
9.4.8 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 (266). If TCustomApplication.StopOnException (275) 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 (266), TCustomApplication.StopOnException (275)

9.4.9 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 (266), TCustomApplication.StopOnException (275)

9.4.10 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 (272) property and properly shut down the application if it is set to True.

When AExitCode is specified, it will passed to System.ExitCode (263), 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 (275) will be used as the value for this argument.

See also: TCustomApplication.Terminated (272), TCustomApplication.Run (267), ExceptionExitCode (275), System.ExitCode (263)
9.4.11  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 (274) property, the search is performed case sensitive or case insensitive.

Options are identified as command-line parameters which start with OptionChar (274) (by default the dash ('-') character).

See also: HasOption (269), GetOptionValue (268), CheckOptions (269), CaseSensitiveOptions (274), OptionChar (274)

9.4.12  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 (274) property, the search is performed case sensitive or case insensitive.

Options are identified as command-line parameters which start with OptionChar (274) (by default the dash ('-') character).

If an option can appear multiple times, use TCustomApplication.GetOptionValues (269) to retrieve all values. This function only returns the value of the first occurrence of an option.

See also: FindOptionIndex (268), HasOption (269), CheckOptions (269), CaseSensitiveOptions (274), OptionChar (274), TCustomApplication.GetOptionValues (269)
9.4.13 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 (268) will only return the first occurrence of a value.

Errors: None.

See also: TCustomApplication.GetOptionValue (268)

9.4.14 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 (274) property, the search is performed case sensitive or case insensitive.

Options are identified as command-line parameters which start with OptionChar (274) (by default the dash ('-') character).

See also: FindOptionIndex (268), GetOptionValue (268), CheckOptions (269), CaseSensitiveOptions (274), OptionChar (274)

9.4.15 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 value 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 (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, GetOptionValue, HasOption, CaseSensitiveOptions, OptionChar

9.4.16 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. 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, TCustomApplication.Checkoptions, TCustomApplication.GetOptionValue, TCustomApplication.GetOptionValues
9.4.17 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 (274)

9.4.18 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 (528)

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 (??)

9.4.19 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 (??)
9.4.20  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 (272)

9.4.21  TCustomApplication.Terminated
Synopsis: Was Terminate called or not

Declaration: Property Terminated : Boolean
   Visibility: public
   Access: Read

Description: Terminated indicates whether Terminate (267) 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 (267)

9.4.22  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 (272)

9.4.23  TCustomApplication.OnException
Synopsis: Exception handling event

Declaration: Property OnException : TExceptionEvent
   Visibility: public
   Access: Read,Write
CHAPTER 9. REFERENCE FOR UNIT 'CUSTAPP'

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 (267). If the event is set, then it is called by the HandleException (266) routine. Do not use the OnException event directly, instead call HandleException.

See also: ShowException (267)

9.4.24 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 (??)

9.4.25 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 (273)

9.4.26 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 (274). It is equivalent to calling ParamStr (??).

See also: ParamCount (274), Paramstr (??)
9.4.27 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 (273) property.

See also: Params (273), Paramstr (?), ParamCount (??)

9.4.28 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 (271) can be used.

See also: GetEnvironmentList (271), TCustomApplication.Params (273)

9.4.29 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 (268), GetOptionValue (268), HasOption (269), CaseSensitiveOptions (274), CheckOptions (269)

9.4.30 TCustomApplication.CaseSensitiveOptions

Synopsis: Are options interpreted case sensitive or not

Declaration: Property CaseSensitiveOptions : Boolean

Visibility: public
Access: Read, Write
Description: CaseSensitiveOptions determines whether FindOptionIndex (268) and CheckOptions (269) 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 (268), GetOptionValue (268), HasOption (269), OptionChar (274), CheckOptions (269)

### 9.4.31 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 (267) and HandleException (266) procedures in case of an unhandled exception in the DoRun code. If StopOnException is True then Terminate (267) will be called after the exception was handled.

See also: Run (267), HandleException (266), Terminate (267)

### 9.4.32 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 (267)

### 9.4.33 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 (271) 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 (271)
9.4.34  **TCustomApplication.SingleInstance**

Synopsis: Single instance used to control single application instance behaviour

Declaration: 

```plaintext
Property SingleInstance : TBaseSingleInstance
```

Visibility: public

Access: Read

Description: SingleInstance is used when TCustomApplication.SingleInstanceEnabled (276) 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 (276), TCustomApplication.SingleInstanceEnabled (276)

9.4.35  **TCustomApplication.SingleInstanceClass**

Synopsis: Class to use when creating single instance

Declaration: 

```plaintext
Property SingleInstanceClass : TBaseSingleInstanceClass
```

Visibility: public

Access: Read, Write

Description: SingleInstanceClass can be used to set the class used to instantiate SingleInstance (276). The default class is determined by the global singleinstance default class as specified in #fcl.singleinstance.DefaultSingleInstanceClass (??).

See also: TCustomApplication.SingleInstance (276), DefaultSingleInstanceClass (??)

9.4.36  **TCustomApplication.SingleInstanceEnabled**

Synopsis: Enable single application instance control.

Declaration: 

```plaintext
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 (276) using TCustomApplication.SingleInstanceClass (276) and starts the check to see whether this application is a client or server instance.

See also: TCustomApplication.SingleInstance (276), TCustomApplication.SingleInstanceClass (276)
Chapter 10

Reference for unit ’daemonapp’

10.1 Used units

Table 10.1: Used units by unit ’daemonapp’

<table>
<thead>
<tr>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Classes</td>
<td>??</td>
</tr>
<tr>
<td>CustApp</td>
<td>263</td>
</tr>
<tr>
<td>eventlog</td>
<td>526</td>
</tr>
<tr>
<td>rtlconsts</td>
<td>??</td>
</tr>
<tr>
<td>System</td>
<td>??</td>
</tr>
<tr>
<td>sysutils</td>
<td>??</td>
</tr>
</tbody>
</table>

10.2 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 (295).

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 (299).

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 (302).

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 (528) 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.

10.3 Constants, types and variables

10.3.1 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

### 10.3.2 Types

TCurrentStatus = (csStopped, csStartPending, csStopPending, csRunning,

    csContinuePending, csPausePending, csPaused)

<table>
<thead>
<tr>
<th>Value</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>csContinuePending</td>
<td>The daemon is continuing, but not yet running</td>
</tr>
<tr>
<td>csPaused</td>
<td>The daemon is paused: running but not active.</td>
</tr>
<tr>
<td>csPausePending</td>
<td>The daemon is about to be paused.</td>
</tr>
<tr>
<td>csRunning</td>
<td>The daemon is running (it is operational).</td>
</tr>
<tr>
<td>csStartPending</td>
<td>The daemon is starting, but not yet fully running.</td>
</tr>
<tr>
<td>csStopped</td>
<td>The daemon is stopped, i.e. inactive.</td>
</tr>
<tr>
<td>csStopPending</td>
<td>The daemon is stopping, but not yet fully stopped.</td>
</tr>
</tbody>
</table>

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 (302) definition.

TCustomDaemonMapperClass = Class of TCustomDaemonMapper

TCustomDaemonMapperClass is the class of TCustomDaemonMapper. It is used in the RegisterDaemonMapper (284) 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 (285) 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 10.3: Enumeration values for type TDaemonOption

<table>
<thead>
<tr>
<th>Value</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>doAllowPause</td>
<td>The daemon can be paused.</td>
</tr>
<tr>
<td>doAllowStop</td>
<td>The daemon can be stopped.</td>
</tr>
<tr>
<td>doInteractive</td>
<td>The daemon interacts with the desktop.</td>
</tr>
</tbody>
</table>

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 10.4: Enumeration values for type TDaemonRunMode

<table>
<thead>
<tr>
<th>Value</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>drmInstall</td>
<td>Daemon install mode (windows only)</td>
</tr>
<tr>
<td>drmRun</td>
<td>Daemon is running normally</td>
</tr>
<tr>
<td>drmUninstall</td>
<td>Daemon uninstal mode (windows only)</td>
</tr>
<tr>
<td>drmUnknown</td>
<td>Unknown mode</td>
</tr>
</tbody>
</table>

TDaemonRunMode indicates in what mode the daemon application (as a whole) is currently running.

TErrorSeverity = (esIgnore, esNormal, esSevere, esCritical)

Table 10.5: Enumeration values for type TErrorSeverity

<table>
<thead>
<tr>
<th>Value</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>esCritical</td>
<td>Error is logged, and startup is stopped if last known good configuration is active, or system is restarted using last known good configuration</td>
</tr>
<tr>
<td>esIgnore</td>
<td>Ignore startup errors</td>
</tr>
<tr>
<td>esNormal</td>
<td>Error is logged, but startup continues</td>
</tr>
<tr>
<td>esSevere</td>
<td>Error is logged, and startup is continued if last known good configuration is active, or system is restarted using last known good configuration</td>
</tr>
</tbody>
</table>

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 10.6: Enumeration values for type TServiceType

<table>
<thead>
<tr>
<th>Value</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>stDevice</td>
<td>Device driver</td>
</tr>
<tr>
<td>stFileSystem</td>
<td>File system driver</td>
</tr>
<tr>
<td>stWin32</td>
<td>Regular win32 service</td>
</tr>
</tbody>
</table>

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 10.7: Enumeration values for type TStartType

<table>
<thead>
<tr>
<th>Value</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>stAuto</td>
<td>Started automatically by service manager during system startup</td>
</tr>
<tr>
<td>stBoot</td>
<td>During system boot</td>
</tr>
<tr>
<td>stDisabled</td>
<td>Service is not started, it is disabled</td>
</tr>
<tr>
<td>stManual</td>
<td>Started manually by the user or other processes.</td>
</tr>
<tr>
<td>stSystem</td>
<td>During load of device drivers</td>
</tr>
</tbody>
</table>

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 10.8: Enumeration values for type TWinControlCode

<table>
<thead>
<tr>
<th>Value</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>wccHardwareProfileChange</td>
<td></td>
</tr>
<tr>
<td>wccNetBindChange</td>
<td></td>
</tr>
<tr>
<td>wccParamChange</td>
<td></td>
</tr>
<tr>
<td>wccPowerEvent</td>
<td></td>
</tr>
<tr>
<td>wccPreShutdown</td>
<td></td>
</tr>
<tr>
<td>wccSessionChange</td>
<td></td>
</tr>
<tr>
<td>wccTimeChange</td>
<td></td>
</tr>
<tr>
<td>wccShutdown</td>
<td></td>
</tr>
<tr>
<td>wccTriggerEvent</td>
<td></td>
</tr>
<tr>
<td>wccUserModeReboot</td>
<td></td>
</tr>
</tbody>
</table>
TWinControlCodes = Set of TWinControlCode

10.3.3 Variables

AppClass : TCustomDaemonApplicationClass

AppClass can be set to the class of a TCustomDaemonApplication (287) descendant. When the Application (283) 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 (302)) is created.

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

Status message

10.4 Procedures and functions

10.4.1 Application

Synopsis: Application instance

Declaration: function Application : TCustomDaemonApplication

Visibility: default

Description: Application is the TCustomDaemonApplication (287) instance used by this application. The instance is created at the first invocation of this function, so it is possible to use RegisterDaemonApplicationClass (284) to register an alternative TCustomDaemonApplication class to run the application.

See also: TCustomDaemonApplication (287), RegisterDaemonApplicationClass (284)

10.4.2 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 (284) exception with message Msg or it formats the message using Fmt and Args.

See also: EDaemon (284)
CHAPTER 10. REFERENCE FOR UNIT ‘DAEMONAPP’

10.4.3 RegisterDaemonApplicationClass

Synopsis: Register alternative TCustomDaemonApplication class.

Declaration: procedure RegisterDaemonApplicationClass

Visibility: default

Description: RegisterDaemonApplicationClass can be used to register an alternative TCustomDaemonApplication (287) descendant which will be used when creating the global Application (283) instance. Only the last registered class pointer will be used.

See also: TCustomDaemonApplication (287), Application (283)

10.4.4 RegisterDaemonClass

Synopsis: Register daemon

Declaration: procedure RegisterDaemonClass(AClass: TCustomDaemonClass)

Visibility: default

Description: RegisterDaemonClass must be called for each TCustomDaemon (285) descendant that is used in the class: the class pointer and class name are used by the TCustomDaemonMapperClass (280) class to create a TCustomDaemon instance when a daemon is required.

See also: TCustomDaemonMapperClass (280), TCustomDaemon (285)

10.4.5 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 daemon-mapper. The daemonmapper will be used only when the application is being run, by the TCustomDaemonApplication (287) code, so registering an alternative mapping class should happen in the initialization section of the application units.

See also: TCustomDaemonApplication (287), TCustomDaemonMapperClass (280)

10.5 EDaemon

10.5.1 Description

EDeamon is the exception class used by all code in the DaemonApp unit.

See also: DaemonError (283)
10.6 TCustomDaemon

10.6.1 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 (295) can be designed to implement the behaviour of the daemon.

See also: TDaemon (295), TDaemonDef (302), TDaemonController (299), TDaemonApplication (299)

10.6.2 Method overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>285</td>
<td>CheckControlMessages</td>
<td></td>
</tr>
<tr>
<td>285</td>
<td>LogMessage</td>
<td>Log a message to the system log</td>
</tr>
<tr>
<td>286</td>
<td>ReportStatus</td>
<td>Report the current status to the operating system</td>
</tr>
</tbody>
</table>

10.6.3 Property overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Properties</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>287</td>
<td>Controller</td>
<td>r</td>
<td>TDaemonController instance controlling this daemon instance</td>
</tr>
<tr>
<td>286</td>
<td>DaemonThread</td>
<td>r</td>
<td>Thread in which daemon is running</td>
</tr>
<tr>
<td>286</td>
<td>Definition</td>
<td>r</td>
<td>The definition used to instantiate this daemon instance</td>
</tr>
<tr>
<td>287</td>
<td>Logger</td>
<td>r</td>
<td>TEventLog instance used to send messages to the system log</td>
</tr>
<tr>
<td>287</td>
<td>Status</td>
<td>rw</td>
<td>Current status of the daemon</td>
</tr>
</tbody>
</table>

10.6.4 TCustomDaemon.CheckControlMessages

Synopsis:

Declaration: procedure CheckControlMessages(Wait: Boolean)

Visibility: public

Description:

10.6.5 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 (528) instance is used to actually send messages to the system log.

The message is sent with an "error" flag (using TEventLog.Error (532)).

Errors: None.

See also: ReportStatus (286)
10.6.6 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 (285)

10.6.7 TCustomDaemon.Definition

Synopsis: The definition used to instantiate this daemon instance

Declaration: Property Definition : TDaemonDef

Visibility: public
Access: Read

Description: Definition is the TDaemonDef (302) 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 (302)

10.6.8 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 (287)
10.6.9 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 (286)

10.6.10 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 (286)

10.6.11 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 (528) instance used to send messages to the system log. It is used by the LogMessage (285) call, but is accessible through the Logger property in case more configurable logging is needed than offered by LogMessage.

See also: LogMessage (285), TEventLog (528)

10.7 TCustomDaemonApplication

10.7.1 Description

TCustomDaemonApplication is a TCustomApplication (264) 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 (295) instances by checking the TCustomDaemonMapperClass (280) instance that contains the daemon maps.

See also: TCustomApplication (264), TCustomDaemonMapperClass (280)
10.7.2 **Method overview**

<table>
<thead>
<tr>
<th>Page</th>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>288</td>
<td>Create</td>
<td>Constructor for the class instance</td>
</tr>
<tr>
<td>289</td>
<td>CreateDaemon</td>
<td>Create daemon instance</td>
</tr>
<tr>
<td>290</td>
<td>CreateForm</td>
<td>Create a component</td>
</tr>
<tr>
<td>288</td>
<td>Destroy</td>
<td>Clean up the TCustomDaemonApplication instance</td>
</tr>
<tr>
<td>289</td>
<td>InstallDaemons</td>
<td>Install all daemons.</td>
</tr>
<tr>
<td>289</td>
<td>RunDaemons</td>
<td>Run all daemons.</td>
</tr>
<tr>
<td>288</td>
<td>ShowException</td>
<td>Show an exception</td>
</tr>
<tr>
<td>290</td>
<td>StopDaemons</td>
<td>Stop all daemons</td>
</tr>
<tr>
<td>290</td>
<td>UnInstallDaemons</td>
<td>Uninstall all daemons</td>
</tr>
</tbody>
</table>

10.7.3 **Property overview**

<table>
<thead>
<tr>
<th>Page</th>
<th>Properties</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>292</td>
<td>AutoRegisterMessageFile</td>
<td>rw</td>
<td>Automatically register the message file</td>
</tr>
<tr>
<td>291</td>
<td>EventLog</td>
<td>r</td>
<td>Event logger instance</td>
</tr>
<tr>
<td>291</td>
<td>GuiHandle</td>
<td>rw</td>
<td>Handle of GUI loop main application window handle</td>
</tr>
<tr>
<td>291</td>
<td>GUIMainLoop</td>
<td>rw</td>
<td>GUI main loop callback</td>
</tr>
<tr>
<td>290</td>
<td>OnRun</td>
<td>rw</td>
<td>Event executed when the daemon is run.</td>
</tr>
<tr>
<td>291</td>
<td>RunMode</td>
<td>r</td>
<td>Application mode</td>
</tr>
</tbody>
</table>

10.7.4 **TCustomDaemonApplication.Create**

Synopsis: Constructor for the class instance

Declaration: constructor Create(AOwner: TComponent); Override

Visibility: public

Description: Constructor for the class instance

10.7.5 **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 (291)

10.7.6 **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.
10.7.7 TCustomDaemonApplication.CreateDaemon

Synopsis: Create daemon instance

Declaration: function CreateDaemon(DaemonDef: TDaemonDef): TCustomDaemon

Visibility: public

Description: CreateDaemon is called whenever a TCustomDaemon instance must be created from a TDaemonDef daemon definition, passed in DaemonDef. It initializes the TCustomDaemon instance, and creates a controller instance of type TDaemonController to control the daemon. Finally, it assigns the created daemon to the TDaemonDef.Instance property.

Errors: In case of an error, an exception may be raised.

See also: TDaemonController, TCustomDaemon, TDaemonDef, TDaemonDef.Instance

10.7.8 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, TCustomDaemonApplication.UnInstallDaemons, TCustomDaemonApplication.RunDaemons

10.7.9 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, TCustomDaemonApplication.RunDaemons, TCustomDaemonApplication.StopDaemons

10.7.10 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, TCustomDaemonApplication.InstallDaemons, TCustomDaemonApplication.StopDaemons
10.7.11 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 (289), TCustomDaemonApplication.InstallDaemons (289), TCustomDaemonApplication.StopDaemons (289)

10.7.12 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.

10.7.13 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 (289)

10.7.14 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 (289), TCustomDaemonApplication.InstallDaemons (289), TCustomDaemonApplication.UnInstallDaemons (290)
10.7.15 **TCustomDaemonApplication.EventLog**

Synopsis: Event logger instance

Declaration: Property EventLog : TEventLog

Visibility: public

Access: Read

Description: EventLog is the TEventLog (528) 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 (528)

10.7.16 **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 (291)

10.7.17 **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 (291) procedure exists, and the application creates and runs a message loop by itself.

See also: GUIMainLoop (291)

10.7.18 **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 (289), RunDaemons (289) or UnInstallDaemons (290) is called.

See also: InstallDaemons (289), RunDaemons (289), UnInstallDaemons (290)
CHAPTER 10. REFERENCE FOR UNIT 'DAEMONAPP'

10.7.19 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.

10.8 TCustomDaemonMapper

10.8.1 Description

The TCustomDaemonMapper class is responsible for mapping a daemon definition to an actual TDaemon instance. It maintains a TDaemonDefs (306) 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 (302) definitions and the actual TDaemon (295) instances that will be used to run the daemons.

The TCustomDaemonMapper class has no support for streaming. The TDaemonMapper (308) class has support for streaming (and hence visual designing).

See also: TDaemon (295), TDaemonDef (302), TDaemonDefs (306), TDaemonMapper (308)

10.8.2 Method overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>292</td>
<td>Create</td>
<td>Create a new instance of TCustomDaemonMapper</td>
</tr>
<tr>
<td>293</td>
<td>Destroy</td>
<td>Clean up and destroy a TCustomDaemonMapper instance.</td>
</tr>
</tbody>
</table>

10.8.3 Property overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Properties</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>293</td>
<td>DaemonDefs</td>
<td>rw</td>
<td>Collection of daemons</td>
</tr>
<tr>
<td>293</td>
<td>OnCreate</td>
<td>rw</td>
<td>Event called when the daemon mapper is created</td>
</tr>
<tr>
<td>294</td>
<td>OnDestroy</td>
<td>rw</td>
<td>Event called when the daemon mapper is freed.</td>
</tr>
<tr>
<td>294</td>
<td>OnInstall</td>
<td>rw</td>
<td>Event called when the daemons are installed</td>
</tr>
<tr>
<td>294</td>
<td>OnRun</td>
<td>rw</td>
<td>Event called when the daemons are executed.</td>
</tr>
<tr>
<td>294</td>
<td>OnUnInstall</td>
<td>rw</td>
<td>Event called when the daemons are uninstalled</td>
</tr>
</tbody>
</table>

10.8.4 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 (306) 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 (306), TCustomDaemonApplication (287), TCustomDaemonMapper.Destroy (293)

10.8.5 TCustomDaemonMapper.Destroy

Synopsis: Clean up and destroy a TCustomDaemonMapper instance.

Declaration: destructor Destroy; Override

Visibility: public

Description: Destroy frees the DaemonDefs (293) collection and calls the inherited destructor.

See also: TDaemonDefs (306), TCustomDaemonMapper.Create (292)

10.8.6 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 (287)

10.8.7 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 (294), OnUnInstall (294), OnCreate (293), OnDestroy (294)
10.8.8 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 event.

See also: OnCreate (293), OnInstall (294), OnUnInstall (294), OnCreate (293)

10.8.9 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 (294), OnUnInstall (294), OnCreate (293), OnDestroy (294)

10.8.10 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 (294), OnInstall (294), OnCreate (293), OnDestroy (294)

10.8.11 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 (294), OnInstall (294), OnCreate (293), OnDestroy (294)
10.9  TDaemon

10.9.1 Description
TDaemon is a TCustomDaemon (285) descendent which is meant for development in a visual environment: it contains event handlers for all major operations. Whenever a TCustomDaemon method is executed, its 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 (286)), which is not the main program thread.

See also: TCustomDaemon (285), TDaemonController (299)

10.9.2 Property overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Properties</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>298</td>
<td>AfterInstall</td>
<td>rw</td>
<td>Called after the daemon was installed</td>
</tr>
<tr>
<td>298</td>
<td>AfterUnInstall</td>
<td>rw</td>
<td>Called after the daemon is uninstalled</td>
</tr>
<tr>
<td>297</td>
<td>BeforeInstall</td>
<td>rw</td>
<td>Called before the daemon will be installed</td>
</tr>
<tr>
<td>298</td>
<td>BeforeUnInstall</td>
<td>rw</td>
<td>Called before the daemon is uninstalled</td>
</tr>
<tr>
<td>295</td>
<td>Definition</td>
<td></td>
<td></td>
</tr>
<tr>
<td>296</td>
<td>OnContinue</td>
<td>rw</td>
<td>Daemon continue</td>
</tr>
<tr>
<td>298</td>
<td>OnControlCode</td>
<td>rw</td>
<td>Called when a control code is received for the daemon</td>
</tr>
<tr>
<td>299</td>
<td>OnControlCodeEvent</td>
<td>rw</td>
<td></td>
</tr>
<tr>
<td>297</td>
<td>OnExecute</td>
<td>rw</td>
<td>Daemon execute event</td>
</tr>
<tr>
<td>296</td>
<td>OnPause</td>
<td>rw</td>
<td>Daemon pause event</td>
</tr>
<tr>
<td>297</td>
<td>OnShutDown</td>
<td>rw</td>
<td>Daemon shutdown</td>
</tr>
<tr>
<td>295</td>
<td>OnStart</td>
<td>rw</td>
<td>Daemon start event</td>
</tr>
<tr>
<td>296</td>
<td>OnStop</td>
<td>rw</td>
<td>Daemon stop event</td>
</tr>
<tr>
<td>295</td>
<td>Status</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

10.9.3 TDaemon.Definition

Declaration: Property Definition :
Visibility: public
Access:

10.9.4 TDaemon.Status

Declaration: Property Status :
Visibility: public
Access:

10.9.5 TDaemon.OnStart

Synopsis: Daemon start event
Declaration: Property OnStart : TDaemonOKEvent
Visibility: published
CHAPTER 10. REFERENCE FOR UNIT ‘DAEMONAPP’

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 (286) 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 (297) event handler, so the event handler can return at once.

See also: TDaemon.OnStop (296), TDaemon.OnExecute (297), TDaemon.OnContinue (296), ReportStatus (286)

10.9.6 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 (286) method.

If a thread was started in the OnExecute (297) event, this is the place where the thread should be stopped.

See also: TDaemon.OnStart (295), TDaemon.OnPause (296), ReportStatus (286)

10.9.7 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 (286) method.

If a thread was started in the OnExecute (297) event, this is the place where the thread’s execution should be suspended.

See also: TDaemon.OnStop (296), TDaemon.OnContinue (296), ReportStatus (286)

10.9.8 TDaemon.OnContinue

Synopsis: Daemon continue

Declaration: Property OnContinue : TDaemonOKEvent

Visibility: published
CHAPTER 10. REFERENCE FOR UNIT 'DAEMONAPP'

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 method.

If a thread was started in the OnExecute event and it was suspended in a OnPause event, this is the place where the thread’s executed should be resumed.

See also: TDaemon.OnStart, TDaemon.OnPause, ReportStatus

10.9.9 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 event, this is the place where the thread’s executed should be stopped or the thread freed from memory.

See also: TDaemon.OnStart, TDaemon.OnPause, ReportStatus

10.9.10 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, TDaemon.OnStop

10.9.11 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 (298), BeforeUnInstall (298), AfterUnInstall (298)

### 10.9.12 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 (297), BeforeUnInstall (298), AfterUnInstall (298)

### 10.9.13 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 (297), AfterInstall (298), AfterUnInstall (298)

### 10.9.14 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 (297), AfterInstall (298), BeforeUnInstall (298)

### 10.9.15 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 (277)
10.9.16 TDaemon.OnControlCodeEvent

Synopsis:

Declaration: Property OnControlCodeEvent : TCustomControlCodeEvEvent

Visibility: published

Access: Read, Write

Description:

10.10 TDaemonApplication

10.10.1 Description

TDaemonApplication is the default TCustomDaemonApplication (287) descendent that is used to run the daemon application. It is possible to register an alternative TCustomDaemonApplication class (using RegisterDaemonApplicationClass (284)) to run the application in a different manner.

See also: TCustomDaemonApplication (287), RegisterDaemonApplicationClass (284)

10.11 TDaemonController

10.11.1 Description

TDaemonController is a class that is used by the TDaemonApplication (299) 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 (285), TDaemonApplication (299)

10.11.2 Method overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>301</td>
<td>Controller</td>
<td>Controller</td>
</tr>
<tr>
<td>300</td>
<td>Create</td>
<td>Create a new instance of the TDaemonController class</td>
</tr>
<tr>
<td>300</td>
<td>Destroy</td>
<td>Free a TDaemonController instance.</td>
</tr>
<tr>
<td>300</td>
<td>Main</td>
<td>Daemon main entry point.</td>
</tr>
<tr>
<td>301</td>
<td>ReportStatus</td>
<td>Report the status to the operating system.</td>
</tr>
<tr>
<td>300</td>
<td>StartService</td>
<td>Start the service</td>
</tr>
</tbody>
</table>

10.11.3 Property overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Properties</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>302</td>
<td>CheckPoint</td>
<td>r</td>
<td>Send checkpoint signal to the operating system</td>
</tr>
<tr>
<td>301</td>
<td>Daemon</td>
<td>r</td>
<td>Daemon instance this controller controls.</td>
</tr>
<tr>
<td>302</td>
<td>LastStatus</td>
<td>r</td>
<td>Last reported status</td>
</tr>
<tr>
<td>301</td>
<td>Params</td>
<td>r</td>
<td>Parameters passed to the daemon</td>
</tr>
</tbody>
</table>
10.11.4  **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 (299) instance, and `AOwner` will be set to the global TDaemonApplication (299) instance.

See also: TDaemonApplication (299), Destroy (300)

10.11.5  **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 (300)

10.11.6  **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 (300)

10.11.7  **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 (309)
10.11.8  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 (298)

10.11.9  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 (286), TDaemonController.LastStatus (302)

10.11.10  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.

10.11.11  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 (300) method.
10.11.12 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 (301)

10.11.13 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 (301)

10.12 TDaemonDef

10.12.1 Description

TDaemonDef contains the definition of a daemon in the application: The name of the daemon, which TCustomDaemon (285) 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 (299), TDaemon (295)

10.12.2 Method overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>303</td>
<td>Create</td>
<td>Create a new TDaemonDef instance</td>
</tr>
<tr>
<td>303</td>
<td>Destroy</td>
<td>Free a TDaemonDef from memory</td>
</tr>
</tbody>
</table>


### 10.12.3 Property overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Properties</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>303</td>
<td>DaemonClass</td>
<td>r</td>
<td>TDaemon class to use for this daemon</td>
</tr>
<tr>
<td>304</td>
<td>DaemonClassName</td>
<td>rw</td>
<td>Name of the TDaemon class to use for this daemon</td>
</tr>
<tr>
<td>304</td>
<td>Description</td>
<td>rw</td>
<td>Description of the daemon</td>
</tr>
<tr>
<td>305</td>
<td>DisplayName</td>
<td>rw</td>
<td>Displayed name of the daemon (service)</td>
</tr>
<tr>
<td>305</td>
<td>Enabled</td>
<td>rw</td>
<td>Is the daemon enabled or not</td>
</tr>
<tr>
<td>304</td>
<td>Instance</td>
<td>rw</td>
<td>Instance of the daemon class</td>
</tr>
<tr>
<td>306</td>
<td>LogStatusReport</td>
<td>rw</td>
<td>Log the status report to the system log</td>
</tr>
<tr>
<td>304</td>
<td>Name</td>
<td>rw</td>
<td>Name of the daemon (service)</td>
</tr>
<tr>
<td>306</td>
<td>OnCreateInstance</td>
<td>rw</td>
<td>Event called when a daemon in instantiated</td>
</tr>
<tr>
<td>305</td>
<td>Options</td>
<td>rw</td>
<td>Service options</td>
</tr>
<tr>
<td>305</td>
<td>RunArguments</td>
<td>rw</td>
<td>Additional command-line arguments when running daemon.</td>
</tr>
<tr>
<td>306</td>
<td>WinBindings</td>
<td>rw</td>
<td>Windows-specific bindings (windows only)</td>
</tr>
</tbody>
</table>

### 10.12.4 TDaemonDef.Create

**Synopsis:** Create a new TDaemonDef instance

**Declaration:**

```delphi
class TDaemonDef

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 (306)

### 10.12.5 TDaemonDef.Destroy

**Synopsis:** Free a TDaemonDef from memory

**Declaration:**

```delphi
destructor Destroy; Override

Visibility: public
```

**Description:**

Destroy removes the TDaemonDef from memory.

### 10.12.6 TDaemonDef.DaemonClass

**Synopsis:** TDaemon class to use for this daemon

**Declaration:**

```delphi
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 its name in DaemonClassName (304).

See also: DaemonClassName (304), TDaemonMapper (308)
10.12.7 TDaemonDef.Instance
Synopsis: Instance of the daemon class

Declaration: Property Instance : TCustomDaemon

Visibility: public
Access: Read,Write

Description: Instance points to the TDaemon (295) instance that is used when the service is in operation at runtime.

See also: TDaemonDef.DaemonClass (303)

10.12.8 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 (287) creates an instance of the daemon.

See also: TDaemonDef.Instance (304), TDaemonDef.DaemonClass (303), RegisterDaemonClass (284), TCustomDaemon (285), TCustomDaemonApplication (287)

10.12.9 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 (305)

10.12.10 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.
10.12.11 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 (304)

10.12.12 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.

10.12.13 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.

10.12.14 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.
10.12.15 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 (314)

10.12.16 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 T Daemon.

10.12.17 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 (286)

10.13 TDaemonDefs

10.13.1 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 (292) instance will create a collection and maintain it.

See also: TCustomDaemonMapper (292), TDaemonDef (302)
CHAPTER 10. REFERENCE FOR UNIT 'DAEMONAPP'

10.13.2 Method overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>307</td>
<td>Create</td>
<td>Create a new instance of a TDaemonDefs collection.</td>
</tr>
<tr>
<td>308</td>
<td>DaemonDefByName</td>
<td>Find and return instance of daemon definition with given name.</td>
</tr>
<tr>
<td>307</td>
<td>FindDaemonDef</td>
<td>Find and return instance of daemon definition with given name.</td>
</tr>
<tr>
<td>307</td>
<td>IndexOfDaemonDef</td>
<td>Return index of daemon definition</td>
</tr>
</tbody>
</table>

10.13.3 Property overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Properties</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>308</td>
<td>Daemons</td>
<td>rw</td>
<td>Indexed access to TDaemonDef instances</td>
</tr>
</tbody>
</table>

10.13.4 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 TCus-
tomDaemonMapper (292) instance will create a collection and maintain it.

See also: TDaemonDef (302)

10.13.5 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 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 (307), TDaemonDefs.DaemonDefByName (308)

10.13.6 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 instance whose name matches DaemonName. If no definition is found, Nil is returned.

See also: TDaemonDefs.IndexOfDaemonDef (307), TDaemonDefs.DaemonDefByName (308)
10.13.7   TDaemonDefs.DaemonDefByName

Synopsis: Find and return instance of daemon definition with given name.

Declaration: function DaemonDefByName(const DaemonName: string) : T DaemonDef

Visibility: public

Description: FindDaemonDef searches the list of daemon definitions and returns the TDaemonDef instance whose name matches DaemonName. If no definition is found, an EDaemon exception is raised.

The FindDaemonDef call does not raise an error, but returns Nil instead.

Errors: If no definition is found, an EDaemon exception is raised.

See also: TDaemonDefs.IndexOfDaemonDef, TDaemonDefs.FindDaemonDef (307)

10.13.8   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 (302)

10.14   TDaemonMapper

10.14.1   Description

TDaemonMapper is a direct descendent of TCustomDaemonMapper, 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 constructor can be used instead.

See also: CreateNew, Create

10.14.2   Method overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>309</td>
<td>Create</td>
<td>Create a new TDaemonMapper instance and initializes it from streamed resources.</td>
</tr>
<tr>
<td>309</td>
<td>CreateNew</td>
<td>Create a new TDaemonMapper instance without initialization</td>
</tr>
</tbody>
</table>
10.14.3 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 (309) 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 (309)

10.14.4 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 (309)

10.15 TDaemonThread

10.15.1 Description

TDaemonThread is the thread in which the daemons in the application are run. Each daemon is run in its own thread.

It should not be necessary to create these threads manually, the TDaemonController (299) class will take care of this.

See also: TDaemonController (299), TDaemon (295)

10.15.2 Method overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>310</td>
<td>CheckControlMessage</td>
<td>Check if a control message has arrived</td>
</tr>
<tr>
<td>311</td>
<td>ContinueDaemon</td>
<td>Continue the daemon</td>
</tr>
<tr>
<td>310</td>
<td>Create</td>
<td>Create a new thread</td>
</tr>
<tr>
<td>310</td>
<td>Execute</td>
<td>Run the daemon</td>
</tr>
<tr>
<td>311</td>
<td>InterrogateDaemon</td>
<td>Report the daemon status</td>
</tr>
<tr>
<td>311</td>
<td>PauseDaemon</td>
<td>Pause the daemon</td>
</tr>
<tr>
<td>311</td>
<td>ShutDownDaemon</td>
<td>Shut down daemon</td>
</tr>
<tr>
<td>310</td>
<td>StopDaemon</td>
<td>Stops the daemon</td>
</tr>
</tbody>
</table>

10.15.3 Property overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Properties</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>312</td>
<td>Daemon</td>
<td>r</td>
<td>Daemon instance</td>
</tr>
</tbody>
</table>
10.15.4 **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 (312)

10.15.5 **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 (295)

10.15.6 **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.

10.15.7 **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 (312), TDaemonThread.PauseDaemon (311), TDaemonThread.ShutDownDaemon (311), TCustomDaemon (285), TThread.Terminate (?)
10.15.8 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 (285) instance, and calling Suspend to suspend the thread. It returns True if the attempt was successful.

See also: TDaemonThread.StopDaemon (310), TDaemonThread.ContinueDaemon (311), TDaemonThread.ShutDownDaemon (311), TCustomDaemon (285), TThread.Suspend (??)

10.15.9 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 (285) instance. It returns True if the attempt was successful.

See also: TDaemonThread.Daemon (312), TDaemonThread.StopDaemon (310), TDaemonThread.PauseDaemon (311), TDaemonThread.ShutDownDaemon (311), TCustomDaemon (285)

10.15.10 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 value is the result from the method in the TCustomDaemon (285) instance. The thread is terminated in this method.

See also: TDaemonThread.StopDaemon (310), TDaemonThread.PauseDaemon (311), TDaemonThread.ContinueDaemon (311), TCustomDaemon (285), TThread.Terminate (??)

10.15.11 TDaemonThread.InterrogateDaemon

Synopsis: Report the daemon status

Declaration: function InterrogateDaemon : Boolean; Virtual

Visibility: public

Description: InterrogateDaemon simply calls TCustomDaemon.ReportStatus (286) for the daemon that is running in this thread. It always returns True.

See also: TCustomDaemon.ReportStatus (286)
10.15.12 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 (295)

10.16 TDependencies

10.16.1 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 (313) instances.

See also: TDependency (313)

10.16.2 Method overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>312</td>
<td>Create</td>
<td>Create a new instance of a TDependencies collection.</td>
</tr>
</tbody>
</table>

10.16.3 Property overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Properties</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>312</td>
<td>Items</td>
<td>rw</td>
<td>Default property override</td>
</tr>
</tbody>
</table>

10.16.4 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.

10.16.5 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 (313).
See also: TDependency (313)
10.17 TDependency

10.17.1 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 (312), TDaemonDef (302)

10.17.2 Method overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>313</td>
<td>Assign</td>
<td>Assign TDependency instance to another</td>
</tr>
</tbody>
</table>

10.17.3 Property overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Properties</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>313</td>
<td>IsGroup</td>
<td>rw</td>
<td>Name refers to a service group</td>
</tr>
<tr>
<td>313</td>
<td>Name</td>
<td>rw</td>
<td>Name of the service</td>
</tr>
</tbody>
</table>

10.17.4 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.

10.17.5 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 (313)

10.17.6 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 (313)
10.18  TWinBindings

10.18.1  Description

TWinBindings contains windows-specific properties for the daemon definition (in TDaemonDef.WinBindings (306)). If the daemon should not run on Windows, then the properties can be ignored.

See also: TDaemonDef (302), TDaemonDef.WinBindings (306)

10.18.2  Method overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>315</td>
<td>Assign</td>
<td>Copies all properties</td>
</tr>
<tr>
<td>314</td>
<td>Create</td>
<td>Create a new TWinBindings instance</td>
</tr>
<tr>
<td>314</td>
<td>Destroy</td>
<td>Remove a TWinBindings instance from memory</td>
</tr>
</tbody>
</table>

10.18.3  Property overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Properties</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>318</td>
<td>AcceptedCodes</td>
<td>rw</td>
<td></td>
</tr>
<tr>
<td>315</td>
<td>Dependencies</td>
<td>rw</td>
<td>Service dependencies</td>
</tr>
<tr>
<td>315</td>
<td>ErrCode</td>
<td>rw</td>
<td>Service specific error code</td>
</tr>
<tr>
<td>317</td>
<td>ErrorSeverity</td>
<td>rw</td>
<td>Error severity in case of startup failure</td>
</tr>
<tr>
<td>316</td>
<td>GroupName</td>
<td>rw</td>
<td>Service group name</td>
</tr>
<tr>
<td>317</td>
<td>IDTag</td>
<td>rw</td>
<td>Location in the service group</td>
</tr>
<tr>
<td>316</td>
<td>Password</td>
<td>rw</td>
<td>Password for service startup</td>
</tr>
<tr>
<td>316</td>
<td>ServiceType</td>
<td>rw</td>
<td>Type of service</td>
</tr>
<tr>
<td>316</td>
<td>StartType</td>
<td>rw</td>
<td>Service startup type</td>
</tr>
<tr>
<td>316</td>
<td>UserName</td>
<td>rw</td>
<td>Username to run service as</td>
</tr>
<tr>
<td>317</td>
<td>WaitHint</td>
<td>rw</td>
<td>Timeout wait hint</td>
</tr>
<tr>
<td>315</td>
<td>Win32ErrCode</td>
<td>rw</td>
<td>General windows error code</td>
</tr>
</tbody>
</table>

10.18.4  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 (302), TDaemonDef.WinBindings (306), TWinBindings.Dependencies (315)

10.18.5  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 (315), TWinBindings.Create (314)
10.18.6 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.

10.18.7 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 (286) to the windows service manager. If it is zero, then the contents of Win32ErrCode (315) are reported. If it is nonzero, then the windows-errorcode is set to ERROR_SERVICE_SPECIFIC_ERROR.

See also: TWinBindings.Win32ErrCode (315)

10.18.8 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 (286) to the windows service manager. It is sent if ErrCode (315) is zero.

See also: ErrCode (315)

10.18.9 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.
10.18.10 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 (313)

10.18.11 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 (316)

10.18.12 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 (316) property.

This property is only used during installation of the service.

See also: Password (316)

10.18.13 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.
10.18.14  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 (286)

10.18.15  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.

10.18.16  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.

10.18.17  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.
10.18.18  TWinBindings.AcceptedCodes

Synopsis:

Declaration: Property AcceptedCodes : TWinControlCodes
  
  Visibility: published
  
  Access: Read, Write

Description:
Chapter 11

Reference for unit ’DB’

11.1 Used units

Table 11.1: Used units by unit ’DB’

<table>
<thead>
<tr>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Classes</td>
<td>??</td>
</tr>
<tr>
<td>FmtBCD</td>
<td>??</td>
</tr>
<tr>
<td>MaskUtils</td>
<td>??</td>
</tr>
<tr>
<td>System</td>
<td>??</td>
</tr>
<tr>
<td>sysutils</td>
<td>??</td>
</tr>
<tr>
<td>Variants</td>
<td>??</td>
</tr>
</tbody>
</table>

11.2 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 (375) representing a set of records from a database, TField (424) which represents the contents of a field in a record, TDataSource (412) which acts as an event distributor on behalf of a dataset and TParams (501) which can be used to parameterize queries. The databases connections themselves are abstracted in the TDatabase (365) class.

11.3 Constants, types and variables

11.3.1 Constants

DefaultFieldClasses contains the TField (424) descendent class to use when a TDataset instance needs to create fields based on the TFieldDefs (455) field definitions when opening the dataset. The entries can be set to create customized TField descendents for certain field datatypes in all datasets.

\[
\text{dsEditModes} = \{\text{dsEdit}, \text{dsInsert}, \text{dsSetKey}\}
\]

\text{dsEditModes} contains the various values of TDataset.State (401) for which the dataset is in edit mode, i.e. states in which it is possible to set field values for that dataset.

\[
\text{dsMaxBufferCount} = \text{MAXINT} \div 8
\]

Maximum data buffers count for dataset

\[
\text{dsMaxStringSize} = 8192
\]

Maximum size of string fields

\[
\text{dsWriteModes} = \{\text{dsEdit}, \text{dsInsert}, \text{dsSetKey}, \text{dsCalcFields}, \text{dsFilter}, \text{dsNewValue}, \text{dsInternalCalc}, \text{dsRefreshFields}\}
\]

\text{dsWriteModes} contains the various values of TDataset.State (401) for which data can be written to the dataset buffer.


\text{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, varOleStr, varOleStr, varOleStr, varInt64, varError, varError, varError, varError, varError, varOleStr, varOleStr, varOleStr, varOleStr, varString, varUnknown, varDispatch, varOleStr, varOleStr, varOleStr, varOleStr, varOleStr)

\text{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 \text{varError} is stored in the variant value.

\[
\text{ftBlobTypes} = \{\text{ftBlob}, \text{ftMemo}, \text{ftGraphic}, \text{ftFmtMemo}, \text{ftParadoxOle}, \text{ftDBaseOle}, \text{ftTypedBinary}, \text{ftOraBlob}, \text{ftOraClob}, \text{ftWideMemo}\}
\]

\text{ftBlobTypes} is a constant containing all blob field data types. It is to be preferred over the TBlobType (322) range, which contains some non-blob types as well.
CHAPTER 11. REFERENCE FOR UNIT 'DB'

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

11.3.2 Types

LargeInt = Int64

Large (64-bit) integer

PBookmarkFlag = ^TBookmarkFlag

PBookmarkFlag is a convenience type, defined for internal use in TDataSet (375) or one of its descendents.

PBufferList = ^TBufferList

PBufferList is a pointer to a structure of type TBufferList (322). 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 11.2: Enumeration values for type TBlobStreamMode

<table>
<thead>
<tr>
<th>Value</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>bmRead</td>
<td>Read blob data</td>
</tr>
<tr>
<td>bmReadWrite</td>
<td>Read and write blob data</td>
</tr>
<tr>
<td>bmWrite</td>
<td>Write blob data</td>
</tr>
</tbody>
</table>

TBlobStreamMode 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 (375) method. It is an opaque type, and should not be used any more, it is superseded by the TBookmarkStr (322) type.

TBookmarkFlag = (bfCurrent, bfBOF, bfEOF, bfInserted)

<table>
<thead>
<tr>
<th>Value</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>bfBOF</td>
<td>First record in the dataset.</td>
</tr>
<tr>
<td>bfCurrent</td>
<td>Buffer used for the current record</td>
</tr>
<tr>
<td>bfEOF</td>
<td>Last record in the dataset</td>
</tr>
<tr>
<td>bfInserted</td>
<td>Buffer used for insert</td>
</tr>
</tbody>
</table>

TBookmarkFlag is used internally by TDataSet (375) and its descendant types to mark the internal memory buffers. It should not be used in end-user applications.

TBookmarkStr = ansistring

TBookmarkStr is the type used by the TDataSet.Bookmark (395) property. It can be used as a string, but should in fact be considered an opaque type.

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 (375) class to manage the memory buffers for the data. It should not be necessary to use this type in end-user applications.

TDataAction = (daFail, daAbort, daRetry)

<table>
<thead>
<tr>
<th>Value</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>daAbort</td>
<td>The operation should be aborted (edits are undone, and an EAbort exception is raised)</td>
</tr>
<tr>
<td>daFail</td>
<td>The operation should fail (an exception will be raised)</td>
</tr>
<tr>
<td>daRetry</td>
<td>Retry the operation.</td>
</tr>
</tbody>
</table>

TDataAction is used by the TDataSetErrorEvent (324) 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 (365) class.

TDataChangeEvent = procedure(Sender: TObject; Field: TField) of object

TDataChangeEvent is the event handler prototype for the TDatasource.OnDataChange (416) 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 11.5: Enumeration values for type TDataEvent

<table>
<thead>
<tr>
<th>Value</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>deCheckBrowseMode</td>
<td>The browse mode is being checked</td>
</tr>
<tr>
<td>deConnectChange</td>
<td>Unused</td>
</tr>
<tr>
<td>deDataSetChange</td>
<td>The dataset property changed</td>
</tr>
<tr>
<td>deDataSetScroll</td>
<td>The dataset scrolled to another record</td>
</tr>
<tr>
<td>deDisabledStateChange</td>
<td>Unused</td>
</tr>
<tr>
<td>deFieldChange</td>
<td>A field value changed</td>
</tr>
<tr>
<td>deFieldListChange</td>
<td>Event sent when the list of fields of a dataset changes</td>
</tr>
<tr>
<td>deFocusControl</td>
<td>Event sent whenever a control connected to a field should be focused</td>
</tr>
<tr>
<td>deLayoutChange</td>
<td>The layout properties of one of the fields changed</td>
</tr>
<tr>
<td>deParentScroll</td>
<td>Unused</td>
</tr>
<tr>
<td>dePropertyChange</td>
<td>Unused</td>
</tr>
<tr>
<td>deReconcileError</td>
<td>Unused</td>
</tr>
<tr>
<td>deRecordChange</td>
<td>The current record changed</td>
</tr>
<tr>
<td>deUpdateRecord</td>
<td>The record is being updated</td>
</tr>
<tr>
<td>deUpdateState</td>
<td>The dataset state is updated</td>
</tr>
</tbody>
</table>

TDataEvent describes the various events that can be sent to TDatasource (412) instances connected to a TDataset (375) 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 (375) 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 (409), TDataSet.OnPostError (411)
and TDataSet.OnDeleteError (409) 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 (322).

TDataSetNotifyEvent = procedure(DataSet: TDataSet) of object

TDataSetNotifyEvent is used in most of the TDataSet (375) 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 11.6: Enumeration values for type TDataSetState

<table>
<thead>
<tr>
<th>Value</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>dsBlockRead</td>
<td>The dataset is open, but no events are transferred to datasources.</td>
</tr>
<tr>
<td>dsBrowse</td>
<td>The dataset is active, and the cursor can be used to navigate the data.</td>
</tr>
<tr>
<td>dsCalcFields</td>
<td>The dataset is calculating it’s calculated fields.</td>
</tr>
<tr>
<td>dsCurValue</td>
<td>The dataset is showing the current values of a record.</td>
</tr>
<tr>
<td>dsEdit</td>
<td>The dataset is in editing mode: the current record can be modified.</td>
</tr>
<tr>
<td>dsFilter</td>
<td>The dataset is filtering records.</td>
</tr>
<tr>
<td>dsInactive</td>
<td>The dataset is not active. No data is available.</td>
</tr>
<tr>
<td>dsInsert</td>
<td>The dataset is in insert mode: the current record is a new record which can be edited.</td>
</tr>
<tr>
<td>dsInternalCalc</td>
<td>The dataset is calculating it’s internally calculated fields.</td>
</tr>
<tr>
<td>dsNewValue</td>
<td>The dataset is showing the new values of a record.</td>
</tr>
<tr>
<td>dsOldValue</td>
<td>The dataset is showing the old values of a record.</td>
</tr>
<tr>
<td>dsOpening</td>
<td>The dataset is currently opening, but is not yet completely open.</td>
</tr>
<tr>
<td>dsRefreshFields</td>
<td>Dataset is refreshing field values from server after an update.</td>
</tr>
<tr>
<td>dsSetKey</td>
<td>The dataset is calculating the primary key.</td>
</tr>
</tbody>
</table>

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 descend-
ents.

TDateTimeAlias = TDateTime
TDateTimeAlias is no longer used.

TDateTimeRec = record
  case TFieldType of
    ftDate: (Date: LongInt);
    ftTime: (Time: LongInt);
    ftDateTime: (DateTime: TDateTimeAlias);
  end;
end

TDateTimeRec was used by older TDataset (375) 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 (418)

TDBTransactionClass = Class of TDBTransaction

TDBTransactionClass is the class pointer for the TDBTransaction (420) class.

TFieldAttribute = (faHiddenCol, faReadonly, faRequired, faLink, faUnNamed, faFixed)

Table 11.7: Enumeration values for type TFieldAttribute

<table>
<thead>
<tr>
<th>Value</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>faFixed</td>
<td>Fixed length field</td>
</tr>
<tr>
<td>faHiddenCol</td>
<td>Field is a hidden column (used to construct a unique key)</td>
</tr>
<tr>
<td>faLink</td>
<td>Field is a link field for other datasets</td>
</tr>
<tr>
<td>faReadonly</td>
<td>Field is read-only</td>
</tr>
<tr>
<td>faRequired</td>
<td>Field is required</td>
</tr>
<tr>
<td>faUnNamed</td>
<td>Field has no original name</td>
</tr>
</tbody>
</table>

TFieldAttribute is used to denote some attributes of a field in a database. It is used in the Attributes (454) property of TFieldDef (450).

TFieldAttributes = Set of TFieldAttribute

TFieldAttributes is used in the TFieldDef.Attributes (454) property to denote additional attributes of the underlying field.
TFieldChars = Set of Char

TFieldChars is a type used in the TField.ValidChars (441) 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 (398) items class.

TFieldDefsClass = Class of TFieldDefs

TFieldDefClass is used to be able to customize the actual TDataset.FieldDefs (398) class used in a TDataset (375) descendent.

TFieldGetTextEvent = procedure(Sender: TField; var aText: string; DisplayText: Boolean) of object

TFieldGetTextEvent is the prototype for the TField.OnGetText (449) 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>
<thead>
<tr>
<th>Value</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>fkCalculated</td>
<td>The field is calculated on the fly.</td>
</tr>
<tr>
<td>fkData</td>
<td>Field represents actual data in the underlying data structure.</td>
</tr>
<tr>
<td>fkInternalCalc</td>
<td>Field is calculated but stored in an underlying buffer.</td>
</tr>
<tr>
<td>fkLookup</td>
<td>The field is a lookup field.</td>
</tr>
</tbody>
</table>

Table 11.8: Enumeration values for type TFieldKind

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 (326) 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 (424) class. Its 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 (401);

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
    , ftVarBytes, ftAutoInc, ftBlob, ftMemo, ftGraphic
    , ftFmtMemo,
    , ftParadoxOle, ftDBaseOle, ftTypedBinary, ftCursor
    , ftFixedChar, ft WideString, ft Largeint, ft ADT, ftArray
    , ftReference, ftDataSet, ft OraBlob, ftOraClob, ftVariant
    , ftInterface, ft IDispatch, ft Guid, ft TimeStamp, ft FMTBcd
    , ftFixedWideChar, ftWideMemo)
Table 11.9: Enumeration values for type TFieldType

<table>
<thead>
<tr>
<th>Value</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>ftADT</td>
<td>ADT value</td>
</tr>
<tr>
<td>ftArray</td>
<td>Array data</td>
</tr>
<tr>
<td>ftAutoInc</td>
<td>Auto-increment integer value (4 bytes)</td>
</tr>
<tr>
<td>ftBCD</td>
<td>Binary Coded Decimal value (DECIMAL and NUMERIC SQL types)</td>
</tr>
<tr>
<td>ftBlob</td>
<td>Binary data value (no type, no size)</td>
</tr>
<tr>
<td>ftBoolean</td>
<td>Boolean value</td>
</tr>
<tr>
<td>ftBytes</td>
<td>Array of bytes value, fixed size (untyped)</td>
</tr>
<tr>
<td>ftCurrency</td>
<td>Currency value (4 decimal points)</td>
</tr>
<tr>
<td>ftCursor</td>
<td>Cursor data value (no size)</td>
</tr>
<tr>
<td>ftDataSet</td>
<td>Dataset data (blob)</td>
</tr>
<tr>
<td>ftDate</td>
<td>Date value</td>
</tr>
<tr>
<td>ftDateTime</td>
<td>Date/Time (timestamp) value</td>
</tr>
<tr>
<td>ftDBaseOle</td>
<td>Paradox OLE field data</td>
</tr>
<tr>
<td>ftFixedChar</td>
<td>Fixed character array (string)</td>
</tr>
<tr>
<td>ftFixedWideChar</td>
<td>Fixed wide character date (2 bytes per character)</td>
</tr>
<tr>
<td>ftFloat</td>
<td>Floating point value (double)</td>
</tr>
<tr>
<td>ftFMTBcd</td>
<td>Formatted BCD (Binary Coded Decimal) value.</td>
</tr>
<tr>
<td>ftFmtMemo</td>
<td>Formatted memo data value (no size)</td>
</tr>
<tr>
<td>ftGraphic</td>
<td>Graphical data value (no size)</td>
</tr>
<tr>
<td>ftGuid</td>
<td>GUID data value</td>
</tr>
<tr>
<td>ftIDispatch</td>
<td>Dispatch data value</td>
</tr>
<tr>
<td>ftInteger</td>
<td>Regular integer value (4 bytes, signed)</td>
</tr>
<tr>
<td>ftInterface</td>
<td>Interface data value</td>
</tr>
<tr>
<td>ftLargeint</td>
<td>Large integer value (8-byte)</td>
</tr>
<tr>
<td>ftMemo</td>
<td>Binary text data (no size)</td>
</tr>
<tr>
<td>ftOraBlob</td>
<td>Oracle BLOB data</td>
</tr>
<tr>
<td>ftOraClob</td>
<td>Oracle CLOB data</td>
</tr>
<tr>
<td>ftParadoxOle</td>
<td>Paradox OLE field data (no size)</td>
</tr>
<tr>
<td>ftReference</td>
<td>Reference data</td>
</tr>
<tr>
<td>ftSmallint</td>
<td>Small integer value (1 byte, signed)</td>
</tr>
<tr>
<td>ftString</td>
<td>String data value (ansistring)</td>
</tr>
<tr>
<td>ftTime</td>
<td>Time value</td>
</tr>
<tr>
<td>ftTimeStamp</td>
<td>Timestamp data value</td>
</tr>
<tr>
<td>ftTypedBinary</td>
<td>Binary typed data (no size)</td>
</tr>
<tr>
<td>ftUnknown</td>
<td>Unknown data type</td>
</tr>
<tr>
<td>ftVarBytes</td>
<td>Array of bytes value, variable size (untyped)</td>
</tr>
<tr>
<td>ftVariant</td>
<td>Variant data value</td>
</tr>
<tr>
<td>ftWideMemo</td>
<td>Widestring memo data</td>
</tr>
<tr>
<td>ftWideString</td>
<td>Widestring (2 bytes per character)</td>
</tr>
<tr>
<td>ftWord</td>
<td>Word-sized value(2 bytes, unsigned)</td>
</tr>
</tbody>
</table>

TFieldType indicates the type of a TField (424) underlying data, in the DataType (438) property.

TFilterOption = (foCaseInsensitive, foNoPartialCompare)
Table 11.10: Enumeration values for type TFilterOption

<table>
<thead>
<tr>
<th>Value</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>foCaseInsensitive</td>
<td>Filter case insensitively.</td>
</tr>
<tr>
<td>foNoPartialCompare</td>
<td>Do not compare values partially, always compare completely.</td>
</tr>
</tbody>
</table>

TFilterOption enumerates the various options available when filtering a dataset. The TFilterOptions (329) set is used in the TDataSet.FilterOptions (402) 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 (402) property. The available values are described in the TFilterOption (328) type.

TFilterRecordEvent = procedure(DataSet: TDataSet; var Accept: Boolean)

of object

TFilterRecordEvent is the prototype for the TDataSet.OnFilterRecord (410) 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 11.11: Enumeration values for type TGetMode

<table>
<thead>
<tr>
<th>Value</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>gmCurrent</td>
<td>Retrieve the current record.</td>
</tr>
<tr>
<td>gmNext</td>
<td>Retrieve the next record.</td>
</tr>
<tr>
<td>gmPrior</td>
<td>Retrieve the previous record.</td>
</tr>
</tbody>
</table>

TGetMode is used internally by TDataSet (375) 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 11.12: Enumeration values for type TGetResult

<table>
<thead>
<tr>
<th>Value</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>grBOF</td>
<td>The beginning of the recordset is reached</td>
</tr>
<tr>
<td>grEOF</td>
<td>The end of the recordset is reached.</td>
</tr>
<tr>
<td>grError</td>
<td>An error occurred</td>
</tr>
<tr>
<td>grOK</td>
<td>The operation was completed successfully</td>
</tr>
</tbody>
</table>
TGetResult is used by descendents of TDataset (375) when they have to communicate the result of the GetRecord operation back to the TDataset record.

TIndexOption = (ixPrimary, ixUnique, ixDescending, ixCaseInsensitive,
                ixExpression, ixNonMaintained)

Table 11.13: Enumeration values for type TIndexOption

<table>
<thead>
<tr>
<th>Value</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>ixCaseInsensitive</td>
<td>The values in the index are sorted case-insensitively</td>
</tr>
<tr>
<td>ixDescending</td>
<td>The values in the index are sorted descending.</td>
</tr>
<tr>
<td>ixExpression</td>
<td>The values in the index are based on a calculated expression.</td>
</tr>
<tr>
<td>ixNonMaintained</td>
<td>The index is non-maintained, i.e. changing the data will not update the index.</td>
</tr>
<tr>
<td>ixPrimary</td>
<td>The index is the primary index for the data</td>
</tr>
<tr>
<td>ixUnique</td>
<td>The index is a unique index, i.e. each index value can occur only once</td>
</tr>
</tbody>
</table>

TIndexOption describes the various properties that an index can have. It is used in the TIndexOptions (330) set type to describe all properties of an index definition as in TIndexDef (470).

TIndexOptions = Set of TIndexOption

TIndexOptions contains the set of properties that an index can have. It is used in the TIndexDef.Options (472) property to describe all properties of an index definition as in TIndexDef (470).

TLocateOption = (loCaseInsensitive, loPartialKey)

Table 11.14: Enumeration values for type TLocateOption

<table>
<thead>
<tr>
<th>Value</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>loCaseInsensitive</td>
<td>Perform a case-insensitive search</td>
</tr>
<tr>
<td>loPartialKey</td>
<td>Accept partial key matches for string fields</td>
</tr>
</tbody>
</table>

TLocateOption is used in the TDataset.Locate (390) 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 (390) 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 (365) 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 (501) collection.

TParamStyle = (psInterbase, psPostgreSQL, psSimulated)

<table>
<thead>
<tr>
<th>Value</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>psInterbase</td>
<td>Parameters are specified by a ? character</td>
</tr>
<tr>
<td>psPostgreSQL</td>
<td>Parameters are specified by a $N character.</td>
</tr>
<tr>
<td>psSimulated</td>
<td>Parameters are specified by a $N character.</td>
</tr>
</tbody>
</table>

TParamStyle denotes the style in which parameters are specified in a query. It is used in the TParams.ParseSQL (504) 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>
<thead>
<tr>
<th>Value</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>ptInput</td>
<td>Input parameter</td>
</tr>
<tr>
<td>ptInputOutput</td>
<td>Input/output parameter</td>
</tr>
<tr>
<td>ptOutput</td>
<td>Output parameter, filled on result</td>
</tr>
<tr>
<td>ptResult</td>
<td>Result parameter</td>
</tr>
<tr>
<td>ptUnknown</td>
<td>Unknown type</td>
</tr>
</tbody>
</table>

TParamType indicates the kind of parameter represented by a TParam (488) 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 (331) values.

TProviderFlag = (pfInUpdate, pfInWhere, pfInKey, pfHidden, pfRefreshOnInsert, pfRefreshOnUpdate)

Table 11.17: Enumeration values for type TProviderFlag

<table>
<thead>
<tr>
<th>Value</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>pfHidden</td>
<td>Field is a key field and used in the WHERE clause of an update statement</td>
</tr>
<tr>
<td>pfInKey</td>
<td>Changes to the field should be propagated to the database.</td>
</tr>
<tr>
<td>pfInUpdate</td>
<td>Field should be used in the WHERE clause of an update statement in case of</td>
</tr>
<tr>
<td></td>
<td>upWhereChanged.</td>
</tr>
<tr>
<td>pfRefreshOnInsert</td>
<td>This field’s value should be refreshed after insert.</td>
</tr>
<tr>
<td>pfRefreshOnUpdate</td>
<td>This field’s value should be refreshed after update.</td>
</tr>
</tbody>
</table>

TProviderFlag describes how the field should be used when applying updates from a dataset to the database. Each field of a TDataset (375) has one or more of these flags.

TProviderFlags = Set of TProviderFlag

TProviderFlags is used for the TField.ProviderFlags (448) 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 11.18: Enumeration values for type TPSCommandType

<table>
<thead>
<tr>
<th>Value</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>ctDDL</td>
<td>SQL DDL statement</td>
</tr>
<tr>
<td>ctDelete</td>
<td>SQL DELETE Statement</td>
</tr>
<tr>
<td>ctInsert</td>
<td>SQL INSERT Statement</td>
</tr>
<tr>
<td>ctQuery</td>
<td>General SQL statement</td>
</tr>
<tr>
<td>ctSelect</td>
<td>SQL SELECT Statement</td>
</tr>
<tr>
<td>ctStoredProc</td>
<td>Stored procedure statement</td>
</tr>
<tr>
<td>ctTable</td>
<td>Table contents (select * from table)</td>
</tr>
<tr>
<td>ctUnknown</td>
<td>Unknown SQL type or not SQL based</td>
</tr>
<tr>
<td>ctUpdate</td>
<td>SQL UPDATE statement</td>
</tr>
</tbody>
</table>
TPSCmdType is used in the IProviderSupport.PSGetCmdType (343) 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 (375) 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 (333)

TResolverResponse = (rrSkip, rrAbort, rrMerge, rrApply, rrIgnore)

Table 11.19: Enumeration values for type TResolverResponse

<table>
<thead>
<tr>
<th>Value</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>rrAbort</td>
<td>Abort the whole update process, no error message is displayed (no EAbort exception raised).</td>
</tr>
<tr>
<td>rrApply</td>
<td>Replace the update with new values applied by the event handler</td>
</tr>
<tr>
<td>rrIgnore</td>
<td>Ignore the error and remove update from change log</td>
</tr>
<tr>
<td>rrMerge</td>
<td>Merge the update with existing changes on the server.</td>
</tr>
<tr>
<td>rrSkip</td>
<td>Skip the current update, leave it in the change log.</td>
</tr>
</tbody>
</table>

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 11.20: Enumeration values for type

<table>
<thead>
<tr>
<th>Value</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>rmCenter</td>
<td>Try to position the cursor in the middle of the buffer</td>
</tr>
<tr>
<td>rmExact</td>
<td>Reposition at exact the same location in the buffer</td>
</tr>
</tbody>
</table>

TResyncMode is used internally by various TDataset (375) navigation and data manipulation methods such as the TDataset.Refresh (393) method when they need to reset the cursor position in the dataset’s buffer.

TSQLParseOption = (spoCreate, spoEscapeSlash, spoEscapeRepeat, spoUseMacro)
Table 11.21: Enumeration values for type TSQLParseOption

<table>
<thead>
<tr>
<th>Value</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>spoCreate</td>
<td>Indicates existing parameters are cleared and re-created, not updated</td>
</tr>
<tr>
<td>spoEscapeRepeat</td>
<td>Causes an escaped character to be repeated</td>
</tr>
<tr>
<td>spoEscapeSlash</td>
<td>Causes the Slash character ('/') to be escaped</td>
</tr>
<tr>
<td>spoUseMacro</td>
<td>Enables macro expansion in a SQL statement</td>
</tr>
</tbody>
</table>

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 (333) 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 11.22: Enumeration values for type TUpdateAction

<table>
<thead>
<tr>
<th>Value</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>uaAbort</td>
<td>The whole update operation should abort</td>
</tr>
<tr>
<td>uaApplied</td>
<td>Consider the update as applied</td>
</tr>
<tr>
<td>uaFail</td>
<td>Update operation should fail</td>
</tr>
<tr>
<td>uaRetry</td>
<td>Retry the update operation</td>
</tr>
<tr>
<td>uaSkip</td>
<td>The update of the current record should be skipped. (but not discarded)</td>
</tr>
</tbody>
</table>

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 (375) 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 11.23: Enumeration values for type TUpdateKind

<table>
<thead>
<tr>
<th>Value</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>ukDelete</td>
<td>Delete a record in the database.</td>
</tr>
<tr>
<td>ukInsert</td>
<td>Insert a new record in the database.</td>
</tr>
<tr>
<td>ukModify</td>
<td>Modify an existing record in the database.</td>
</tr>
</tbody>
</table>

TUpdateKind indicates what kind of update operation is in progress when applying updates.

TUpdateMode = (upWhereAll, upWhereChanged, upWhereKeyOnly)

Table 11.24: Enumeration values for type TUpdateMode

<table>
<thead>
<tr>
<th>Value</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>upWhereAll</td>
<td>Use all old field values</td>
</tr>
<tr>
<td>upWhereChanged</td>
<td>Use only old field values of modified fields</td>
</tr>
<tr>
<td>upWhereKeyOnly</td>
<td>Only use key fields in the where clause.</td>
</tr>
</tbody>
</table>

TUpdateMode determines how the WHERE clause of update queries for SQL databases should be constructed.

TUpdateStatus = (usUnmodified, usModified, usInserted, usDeleted)

Table 11.25: Enumeration values for type TUpdateStatus

<table>
<thead>
<tr>
<th>Value</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>usDeleted</td>
<td>Record exists in the database, but is locally deleted.</td>
</tr>
<tr>
<td>usInserted</td>
<td>Record does not yet exist in the database, but is locally inserted</td>
</tr>
<tr>
<td>usModified</td>
<td>Record exists in the database but is locally modified</td>
</tr>
<tr>
<td>usUnmodified</td>
<td>Record is unmodified</td>
</tr>
</tbody>
</table>

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 (335) values.

11.3.3 Variables

LoginDialogExProc : function(const ADatabaseName: string; var AUserName : string; var APassword: string; UserNameReadOnly: Boolean = Nil ) : 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.
CHAPTER 11. REFERENCE FOR UNIT 'DB'

11.4 Procedures and functions

11.4.1 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 (??)

11.4.2 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 (338) exception, passing it Msg. If Comp is specified, the name of the component is prepended to the message.

See also: DatabaseErrorFmt (336), EDatabaseError (338)

11.4.3 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);

Visibility: default

Description: DatabaseErrorFmt raises an EDatabaseError (338) 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 (336), EDatabaseError (338)
11.4.4 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 (327), TDateTimeRec (325), DateTimeToDateTimeRec (337)

11.4.5 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 (327), TDateTimeRec (325), DateTimeRecToDateTime (337)

11.4.6 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 (?)

11.4.7 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 (411) as an enumerator for a TDataset (375)

See also: TDataSetEnumerator (411), TDataset (375)
11.4.8 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 (458)

11.4.9 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 (504)

11.5 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.

11.6 EDatabaseError

11.6.1 Description

EDatabaseError is the base class from which database-related exception classes should derive. It is raised by the DatabaseError (336) call.

See also: DatabaseError (336), DatabaseErrorFmt (336)
11.7 EUpdateError

11.7.1 Description

EupdateError is an exception used by the TProvider database support. It should never be raised directly.

See also: EDatabaseError (338)

11.7.2 Method overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>339</td>
<td>Create</td>
<td>Create a new EUpdateError instance</td>
</tr>
<tr>
<td>339</td>
<td>Destroy</td>
<td>Free the EupdateError instance</td>
</tr>
</tbody>
</table>

11.7.3 Property overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Properties</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>340</td>
<td>Context</td>
<td>r</td>
<td>Context in which exception occurred.</td>
</tr>
<tr>
<td>340</td>
<td>ErrorCode</td>
<td>r</td>
<td>Numerical error code.</td>
</tr>
<tr>
<td>340</td>
<td>OriginalException</td>
<td>r</td>
<td>Originally raised exception</td>
</tr>
<tr>
<td>340</td>
<td>PreviousError</td>
<td>r</td>
<td>Previous error number</td>
</tr>
</tbody>
</table>

11.7.4 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, ErrorCode 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 (338)

11.7.5 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 (340)
11.7.6 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 (340), EUpdateError.ErrorCode (340), EUpdateError.PreviousError (340)

11.7.7 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 (340), EUpdateError.Context (340), EUpdateError.PreviousError (340)

11.7.8 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 (338)

11.7.9 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 (340), EUpdateError.Context (340), EUpdateError.OriginalException (340)
CHAPTER 11. REFERENCE FOR UNIT 'DB'

11.8 IProviderSupport

11.8.1 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 (375) 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 (375)

11.8.2 Method overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>341</td>
<td>PSEndTransaction</td>
<td>End an active transaction</td>
</tr>
<tr>
<td>341</td>
<td>PSEexecute</td>
<td>Execute the current command-text.</td>
</tr>
<tr>
<td>342</td>
<td>PSEexecuteStatement</td>
<td>Execute a SQL statement.</td>
</tr>
<tr>
<td>342</td>
<td>PSGetAttributes</td>
<td>Get a list of attributes (metadata)</td>
</tr>
<tr>
<td>342</td>
<td>PSGetCommandText</td>
<td>Return the SQL command executed for getting data.</td>
</tr>
<tr>
<td>343</td>
<td>PSGetCommandType</td>
<td>Return SQL command type</td>
</tr>
<tr>
<td>343</td>
<td>PSGetDefaultOrder</td>
<td>Default order index definition</td>
</tr>
<tr>
<td>343</td>
<td>PSGetIndexDefs</td>
<td>Return a list of index definitions</td>
</tr>
<tr>
<td>343</td>
<td>PSGetKeyFields</td>
<td>Return a list of key fields in the dataset</td>
</tr>
<tr>
<td>343</td>
<td>PSGetParams</td>
<td>Get the parameters in the commandtext</td>
</tr>
<tr>
<td>343</td>
<td>PSGetQuoteChar</td>
<td>Quote character for quoted strings</td>
</tr>
<tr>
<td>343</td>
<td>PSGetTableName</td>
<td>Name of database table which must be updated</td>
</tr>
<tr>
<td>344</td>
<td>PSGetUpdateException</td>
<td>Transform exception to UpdateError</td>
</tr>
<tr>
<td>345</td>
<td>PSInTransaction</td>
<td>Is the dataset in an active transaction.</td>
</tr>
<tr>
<td>345</td>
<td>PSIIsSQLBased</td>
<td>Is the dataset SQL based</td>
</tr>
<tr>
<td>345</td>
<td>PSIIsSQLSupported</td>
<td>Can the dataset support SQL statements</td>
</tr>
<tr>
<td>345</td>
<td>PSReset</td>
<td>Position the dataset on the first record</td>
</tr>
<tr>
<td>346</td>
<td>PSSetCommandText</td>
<td>Set the command-text of the dataset</td>
</tr>
<tr>
<td>346</td>
<td>PSSetParams</td>
<td>Set the parameters for the command text</td>
</tr>
<tr>
<td>346</td>
<td>PSStartTransaction</td>
<td>Start a new transaction</td>
</tr>
<tr>
<td>346</td>
<td>PSUpdateRecord</td>
<td>Update a record</td>
</tr>
</tbody>
</table>

11.8.3 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 (319) returns True). If ACommit is True then the transaction is committed, else it is rolled back.

See also: PSInTransaction (319), PSStartTransaction (319)

11.8.4 IProviderSupport.PSEexecute

Synopsis: Execute the current command-text.

Declaration: procedure PSEexecute

341
Visibility: default

Description: PSExecute executes the current SQL statement: the command as it is returned by PSGetCommandText (319).

See also: PSGetCommandText (319), PSEexecuteStatement (319)

### 11.8.5 IProviderSupport.PSEexecuteStatement

**Synopsis:** Execute a SQL statement.

**Declaration:**
```pascal
function PSEexecuteStatement(const ASQL: string; AParams: TParams; 
                           ResultSet: Pointer) : Integer
```

Visibility: default

Description: PSEexecuteStatement 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 successfully.

PSEexecuteStatement does not modify the content of CommandText: PSGetCommandText (319) returns the same value before and after a call to PSEexecuteStatement.

See also: PSGetCommandText (319), PSSetCommandText (319), PSEexecuteStatement (319)

### 11.8.6 IProviderSupport.PSGetAttributes

**Synopsis:** Get a list of attributes (metadata)

**Declaration:**
```pascal
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 (319)

### 11.8.7 IProviderSupport.PSGetCommandText

**Synopsis:** Return the SQL command executed for getting data.

**Declaration:**
```pascal
function PSGetCommandText : string
```

Visibility: default

Description: PSGetCommandText returns the SQL command that is executed when the PSEexecute (319) function is called (for a TSQLQuery this would be the SQL property) or when the dataset is opened.

See also: PSEexecute (319), PSSetCommandText (319)
11.8.8 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 (319)). The list of possible command types is enumerated in TPSCommandType (332).

See also: PSGetCommandText (319), TPSCommandType (332), PExecute (319)

11.8.9 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 (319)) that represents the default sort order.

See also: PSGetIndexDefs (319), PSGetKeyFields (319)

11.8.10 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 (319), PSGetKeyFields (319)

11.8.11 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 (424) property.

See also: PSGetIndexDefs (319), PSGetDefaultOrder (319), TField.ProviderOptions (424), TProviderFlags (332)
11.8.12 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 (319)). This is usually the Params property of a TDataset (375) descendant.

See also: PSGetCommandText (319), PSSetParams (319)

11.8.13 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 (319)

11.8.14 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 (319)

11.8.15 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 (339) exception. The previous EUpdateError exception in the update batch is passed in Prev.

See also: EUpdateError (339)
11.8.16 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 (319), PSStartTransaction (319)

11.8.17 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 (319) which indicates whether SQL statements can be executed using PSEexecuteCommand (319)

See also: PSIsSQLSupported (319), PSEexecuteCommand (319)

11.8.18 IProviderSupport.PSIsSQLSupported

Synopsis: Can the dataset support SQL statements

Declaration: function PSIsSQLSupported : Boolean

Visibility: default

Description: PSIsSQLSupported returns True if PSEexecuteCommand (319) can be used to execute SQL statements on the underlying database.

See also: PSEexecuteCommand (319)

11.8.19 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 (387), TDataset.Open (392)
11.8.20  IProviderSupport.PSSetCommandText
Synopsis: Set the command-text of the dataset

Declaration: procedure PSSetCommandText(const CommandText: string)

Visibility: default

Description: PSSetCommandText sets the command text (SQL) statement that is executed by PSEexecute or that is used to open the dataset.

See also: PSEexecute (319), PSGetCommandText (319), PSSetParams (319)

11.8.21  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 (319), PSGetParams (319)

11.8.22  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 (319) returned False).

See also: PSEndTransaction (319), PSIntransaction (319)

11.8.23  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: PSIIsSQLSupported (319), PSEexecuteCommand (319)
11.9  TAutoIncField

11.9.1  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 (424)

11.9.2  Method overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>347</td>
<td>Create</td>
<td>Create a new instance of the TAutoIncField class.</td>
</tr>
</tbody>
</table>

11.9.3  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 (424) class’ fields.

See also: TField (424)

11.10  TBCDField

11.10.1  Description
TBCDField is the class used when a dataset must manage data of Binary Coded Decimal type.
(TField.DataType (438) equals ftBCD). It initializes some of the properties of the TField (424)
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 (466) 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 (375), TField (424), TFMTBCDField (466)

11.10.2  Method overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>348</td>
<td>CheckRange</td>
<td>Check whether a value falls within the allowed range</td>
</tr>
<tr>
<td>348</td>
<td>Create</td>
<td>Create a new instance of a TBCDField class.</td>
</tr>
</tbody>
</table>
CHAPTER 11. REFERENCE FOR UNIT 'DB'

11.10.3 Property overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Properties</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>349</td>
<td>Currency</td>
<td>rw</td>
<td>Does the field represent a currency amount</td>
</tr>
<tr>
<td>349</td>
<td>MaxValue</td>
<td>rw</td>
<td>Maximum value for the field</td>
</tr>
<tr>
<td>350</td>
<td>MinValue</td>
<td>rw</td>
<td>Minimum value for the field</td>
</tr>
<tr>
<td>349</td>
<td>Precision</td>
<td>rw</td>
<td>Precision of the BCD field</td>
</tr>
<tr>
<td>350</td>
<td>Size</td>
<td></td>
<td>Number of decimals after the decimal separator</td>
</tr>
<tr>
<td>348</td>
<td>Value</td>
<td>rw</td>
<td>Value of the field contents as a Currency type</td>
</tr>
</tbody>
</table>

11.10.4 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 (424) properties to configure the instance for working with BCD data values.

See also: TField (424)

11.10.5 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 (350) and MaxValue (349) properties. If the value lies outside of the allowed range, then False is returned.

See also: MaxValue (349), MinValue (350)

11.10.6 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 (442) property to a currency type field. It returns the same value as the TField.AsCurrency (432) field.

See also: TField.Value (442), TField.AsCurrency (432)

348
11.10.7  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 (350), 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 (350), TBCDField.Value (348)

11.10.8  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 (438) and TField.Text (441) 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 (438), TField.Text (441)

11.10.9  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 (350) should also be set, because it will also be checked.

See also: TBCDField.MinValue (350), TBCDField.CheckRange (348)
11.10.10 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 (349) should also be set, because it will also be checked.

See also: TBCDField.MaxValue (349), TBCDField.CheckRange (348)

11.10.11 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 (349) field.

See also: TBCDField.Precision (349)

11.11 TBinaryField

11.11.1 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 (424) 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 (357) or TVarBytesField (512) should be created.

See also: TDataset (375), TField (424), TBytesField (357), TVarBytesField (512)

11.11.2 Method overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>351</td>
<td>Create</td>
<td>Create a new instance of a TBinaryField class.</td>
</tr>
</tbody>
</table>

11.11.3 Property overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Properties</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>351</td>
<td>Size</td>
<td></td>
<td>Size of the binary data</td>
</tr>
</tbody>
</table>
11.11.4  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 (424)

11.11.5  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 (441)

11.12  TBlobField

11.12.1  Description

TBlobField is the class used when a dataset must manage BLOB data. (TField.DataType (438) equals ftBLOB). It initializes some of the properties of the TField (424) 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 (485), TWideMemoField (513) or TGraphicField (469). 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 (375), TField (424), TMemoField (485), TWideMemoField (513), TGraphicField (469)

11.12.2  Method overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>352</td>
<td>Clear</td>
<td>Clear the BLOB field’s contents</td>
</tr>
<tr>
<td>352</td>
<td>Create</td>
<td>Create a new instance of a TBlobField class.</td>
</tr>
<tr>
<td>352</td>
<td>IsBlob</td>
<td>Is the field a blob field</td>
</tr>
<tr>
<td>352</td>
<td>LoadFromFile</td>
<td>Load the contents of the field from a file</td>
</tr>
<tr>
<td>353</td>
<td>LoadFromStream</td>
<td>Load the field’s contents from stream</td>
</tr>
<tr>
<td>353</td>
<td>SaveToFile</td>
<td>Save field contents to a file</td>
</tr>
<tr>
<td>353</td>
<td>SaveToStream</td>
<td>Save the field’s contents to stream</td>
</tr>
<tr>
<td>354</td>
<td>SetFieldType</td>
<td>Set field type</td>
</tr>
</tbody>
</table>
11.12.3 Property overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Properties</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>354</td>
<td>BlobSize</td>
<td>r</td>
<td>Size of the current blob</td>
</tr>
<tr>
<td>355</td>
<td>BlobType</td>
<td>rw</td>
<td>Type of blob</td>
</tr>
<tr>
<td>354</td>
<td>Modified</td>
<td>rw</td>
<td>Has the field’s contents been modified.</td>
</tr>
<tr>
<td>355</td>
<td>Size</td>
<td></td>
<td>Size of the blob field</td>
</tr>
<tr>
<td>355</td>
<td>Transliterate</td>
<td>rw</td>
<td>Should the contents of the field be transliterated</td>
</tr>
<tr>
<td>354</td>
<td>Value</td>
<td>rw</td>
<td>Return the field’s contents as a string</td>
</tr>
</tbody>
</table>

11.12.4 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 (424) properties to configure the instance for working with BLOB data.

See also: TField (424)

11.12.5 TBlobField.Clear

Synopsis: Clear the BLOB field’s contents

Declaration: procedure Clear; Override

Visibility: public

Description: Clear overrides the TField implementation of TField.Clear (429). It creates and immediately releases an empty blob stream in write mode, effectively clearing the contents of the BLOB field.

See also: TField.Clear (429), TField.IsNull (440)

11.12.6 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 (430)

11.12.7 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, and then calls LoadFromStream (353) 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 not available for reading, an exception will be raised.

See also: LoadFromStream (353), SaveToFile (353)

11.12.8 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 (??) descendant. 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 (353), LoadFromFile (352)

11.12.9 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, and then calls SaveToStream (353) 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 (352), SaveToStream (353)

11.12.10 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 (??) descendant. 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 (353), LoadFromStream (353)
11.12.11 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 (438)

11.12.12 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 (441), TField.DataSize (438)

11.12.13 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 (353)

11.12.14 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 (434) property.

See also: TField.Value (442), TField.AsString (434)
11.12.15 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 (394) 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 (510), TDataset.Translate (394)

11.12.16 TBlobField.BlobType

Synopsis: Type of blob

Declaration: Property BlobType : TBlobType

Visibility: published
Access: Read, Write

Description: BlobType is an alias for TField.DataType (438), but with a restricted set of values. Setting BlobType is equivalent to setting the TField.DataType (438) property.

See also: TField.DataType (438)

11.12.17 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 (354) property instead.

See also: BlobSize (354)

11.13 TBooleanField

11.13.1 Description

TBooleanField is the field class used by TDataset (375) whenever it needs to manage boolean data (TField.DataType (438) equals ftBoolean). It overrides some properties and methods of TField (424) 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 (375), TField (424)
11.13.2 Method overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>356</td>
<td>Create</td>
<td>Create a new instance of the TBooleanField class.</td>
</tr>
</tbody>
</table>

11.13.3 Property overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Properties</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>356</td>
<td>DisplayValues</td>
<td>rw</td>
<td>Textual representation of the true and false values</td>
</tr>
<tr>
<td></td>
<td>Value</td>
<td>rw</td>
<td>Value of the field as a boolean value</td>
</tr>
</tbody>
</table>

11.13.4 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 (424) properties to configure it for working with boolean values.

See also: TField (424)

11.13.5 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 (442) by TBooleanField as a boolean value. It returns the same value as the TField.AsBoolean (432) property.

See also: TField.AsBoolean (432), TField.Value (442)

11.13.6 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.
CHAPTER 11. REFERENCE FOR UNIT 'DB'

11.14  TBytesField

11.14.1  Description

TBytesField is the class used when a dataset must manage data of fixed-size binary type. (TField.DataType (438) equals ftBytes). It initializes some of the properties of the TField (424) 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 (375), TField (424), TVarBytesField (512)

11.14.2  Method overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>357</td>
<td>Create</td>
<td>Create a new instance of a TBytesField class.</td>
</tr>
</tbody>
</table>

11.14.3  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 (424) properties to configure the instance for working with binary data values.

See also: TField (424)

11.15  TCheckConstraint

11.15.1  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 (359) property, or a constraint enforced by the user on a particular dataset instance stored in TCheckConstraint.CustomConstraint (358)

See also: TCheckConstraints (359), TCheckConstraint.ImportedConstraint (359), TCheckConstraint.CustomConstraint (358)

11.15.2  Method overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>358</td>
<td>Assign</td>
<td>Assign one constraint to another</td>
</tr>
</tbody>
</table>

See also: TField.AsString (434), TField.Text (441)
11.15.3 Property overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Properties</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>358</td>
<td>CustomConstraint</td>
<td>rw</td>
<td>User-defined constraint</td>
</tr>
<tr>
<td>358</td>
<td>ErrorMessage</td>
<td>rw</td>
<td>Message to display when the constraint is violated</td>
</tr>
<tr>
<td>359</td>
<td>FromDictionary</td>
<td>rw</td>
<td>True if the constraint is imported from a datadictionary</td>
</tr>
<tr>
<td>359</td>
<td>ImportedConstraint</td>
<td>rw</td>
<td>Constraint imported from the database engine</td>
</tr>
</tbody>
</table>

11.15.4 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 (359), TCheckConstraint.CustomConstraint (358)

11.15.5 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 (375) 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 (358)

See also: TCheckConstraint.ErrorMessage (358)

11.15.6 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 (358)
11.15.7 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 (375) descendents to indicate the source of the constraint, but is otherwise ignored.

See also: TCheckConstraint.ImportedConstraint (359)

11.15.8 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 (375) descendent.

See also: TCheckConstraint.CustomConstraint (358)

11.16 TCheckConstraints

11.16.1 Description

TCheckConstraints is a TCollection descendent which keeps a collection of TCheckConstraint (357) items. It overrides the Add (360) method to return a TCheckConstraint instance.

See also: TCheckConstraint (357)

11.16.2 Method overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>360</td>
<td>Add</td>
<td>Add new TCheckConstraint item to the collection</td>
</tr>
<tr>
<td>360</td>
<td>Create</td>
<td>Create a new instance of the TCheckConstraints class.</td>
</tr>
</tbody>
</table>

11.16.3 Property overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Properties</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>360</td>
<td>Items</td>
<td>rw</td>
<td>Indexed access to the items in the collection</td>
</tr>
</tbody>
</table>
11.16.4 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 (375) 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 (357) class pointer.

See also: TCheckConstraint (357), TDataset (375)

11.16.5 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 (357) instance to the collection. It returns the newly added instance.

See also: TCheckConstraint (357), #rtl.classes.TCollection.Add (??)

11.16.6 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 (??)

11.17 TCurrencyField

11.17.1 Description

TCurrencyField is the field class used by TDataset (375) when it needs to manage currency-valued data. (TField.Datatype (438) equals ftCurrency). It simply sets some TField (424) 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 (424), TDataset (375)
11.17.2 Method overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>361</td>
<td>Create</td>
<td>Create a new instance of a TCurrencyField.</td>
</tr>
</tbody>
</table>

11.17.3 Property overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Properties</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>361</td>
<td>Currency</td>
<td></td>
<td>Is the field a currency field</td>
</tr>
</tbody>
</table>

11.17.4 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 (361)) to be able to work with currency data.

**See also:** TField (424), TCurrencyField.Currency (361)

11.17.5 TCurrencyField.Currency

**Synopsis:** Is the field a currency field

**Declaration:**

```
Property Currency :
```

**Visibility:** published

**Access:**

**Description:** `Currency` is inherited from TFloatField.Currency (465) 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 (465)

11.18 TCustomConnection

11.18.1 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 (362)) and to deactivate the connection (TCustomConnection.Close (362)), plus a property to inspect the state (Connected (363)) of the connected.

**See also:** TCustomConnection.Open (362), TCustomConnection.Close (362), TCustomConnection.Connected (363)

11.18.2 Method overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>362</td>
<td>Close</td>
<td>Close the connection</td>
</tr>
<tr>
<td>362</td>
<td>Destroy</td>
<td>Remove the TCustomconnection instance from memory</td>
</tr>
<tr>
<td>362</td>
<td>Open</td>
<td>Makes the connection to the server</td>
</tr>
</tbody>
</table>
11.18.3 Property overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Properties</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>364</td>
<td>AfterConnect</td>
<td>rw</td>
<td>Event triggered after a connection is made.</td>
</tr>
<tr>
<td>364</td>
<td>AfterDisconnect</td>
<td>rw</td>
<td>Event triggered after a connection is closed.</td>
</tr>
<tr>
<td>365</td>
<td>BeforeConnect</td>
<td>rw</td>
<td>Event triggered before a connection is made.</td>
</tr>
<tr>
<td>365</td>
<td>BeforeDisconnect</td>
<td>rw</td>
<td>Event triggered before a connection is closed.</td>
</tr>
<tr>
<td>363</td>
<td>Connected</td>
<td>rw</td>
<td>Is the connection established or not</td>
</tr>
<tr>
<td>363</td>
<td>DataSetCount</td>
<td>r</td>
<td>Number of datasets connected to this connection</td>
</tr>
<tr>
<td>363</td>
<td>DataSets</td>
<td>r</td>
<td>Datasets linked to this connection</td>
</tr>
<tr>
<td>364</td>
<td>LoginPrompt</td>
<td>rw</td>
<td>Should the OnLogin be triggered</td>
</tr>
<tr>
<td>365</td>
<td>OnLogin</td>
<td>rw</td>
<td>Event triggered when a login prompt is shown.</td>
</tr>
</tbody>
</table>

11.18.4 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` (365) 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` (364) event is triggered.

Calling the `Close` method is equivalent to setting the `Connected` (363) 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` (338) exception will be raised.

See also: TCustomConnection.BeforeDisconnect (365), TCustomConnection.AfterDisconnect (364), TCustomConnection.Open (362), TCustomConnection.Connected (363)

11.18.5 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 (362)

11.18.6 TCustomConnection.Open

Synopsis: Makes the connection to the server

Declaration: `procedure Open`
CHAPTER 11. REFERENCE FOR UNIT 'DB'

Visibility: public

Description: Open establishes the connection with the server if it was not yet connected. Calling this method first triggers the BeforeConnect (365) event. If an exception is raised during the execution of that event handler, the connect process is aborted. If LoginPrompt (364) is True, the OnLogin (365) event handler is called. Only after this event, the connection is actually established. After the connection was established, the AfterConnect (364) event is triggered.

Calling the Open method is equivalent to setting the Connected (363) 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 (365), TCustomConnection.LoginPrompt (364), TCustomConnection.OnLogin (365), TCustomConnection.AfterConnect (364), TCustomConnection.Connected (363)

### 11.18.7 TCustomConnectionDataSetCount

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 (363) array property. As implemented in TCustomConnection, this property is always zero. Descendent classes implement the actual count.

See also: TDataSet (375), TCustomConnection.Datasets (363)

### 11.18.8 TCustomConnectionDataSets

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 (363).

See also: DatasetCount (363)

### 11.18.9 TCustomConnectionConnected

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 (362), or to False to break it (equivalent to calling TCustomConnection.Close (362)).

See also: TCustomConnection.Open (362), TCustomConnection.Close (362)

**11.18.10 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 (365)

**11.18.11 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 (362). It can be used to open datasets, or indicate a connection status change.

See also: TCustomConnection.Open (362), TCustomConnection.BeforeConnect (365), TCustomConnection.OnLogin (365)

**11.18.12 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 (362). It can be used for instance to indicate a connection status change.

See also: TCustomConnection.Close (362), TCustomConnection.BeforeDisconnect (365)
11.18.13 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 (362). 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 (362), TCustomConnection.AfterConnect (364), TCustomConnection.OnLogin (365)

11.18.14 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 (362). 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 (362), TCustomConnection.AfterDisconnect (364)

11.18.15 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 (364) property is True, after the TCustomConnection.BeforeConnect (365) event, but before the connection is actually established.

See also: TCustomConnection.BeforeConnect (365), TCustomConnection.LoginPrompt (364), TCustomConnection.Open (362)

11.19 TDatabase

11.19.1 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 (375), 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 (369) 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 descendant classes.

See also: TDataset (375), TDatabase (365)

### 11.19.2 Method overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>367</td>
<td>CloseDataSets</td>
<td>Close all connected datasets</td>
</tr>
<tr>
<td>367</td>
<td>CloseTransactions</td>
<td>End all transactions</td>
</tr>
<tr>
<td>366</td>
<td>Create</td>
<td>Initialize a new TDatabase class instance</td>
</tr>
<tr>
<td>366</td>
<td>Destroy</td>
<td>Remove a TDatabase instance from memory</td>
</tr>
<tr>
<td>367</td>
<td>EndTransaction</td>
<td>End an active transaction</td>
</tr>
<tr>
<td>367</td>
<td>StartTransaction</td>
<td>Start a new transaction</td>
</tr>
</tbody>
</table>

### 11.19.3 Property overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Properties</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>369</td>
<td>Connected</td>
<td>rw</td>
<td>Is the database connected</td>
</tr>
<tr>
<td>369</td>
<td>DatabaseName</td>
<td>rw</td>
<td>Database name or path</td>
</tr>
<tr>
<td>368</td>
<td>Directory</td>
<td>rw</td>
<td>Directory for the database</td>
</tr>
<tr>
<td>368</td>
<td>IsSQLBased</td>
<td>r</td>
<td>Is the database SQL based.</td>
</tr>
<tr>
<td>369</td>
<td>KeepConnection</td>
<td>rw</td>
<td>Should the connection be kept active</td>
</tr>
<tr>
<td>369</td>
<td>Params</td>
<td>rw</td>
<td>Connection parameters</td>
</tr>
<tr>
<td>368</td>
<td>TransactionCount</td>
<td>r</td>
<td>Number of transaction components connected to this database.</td>
</tr>
<tr>
<td>368</td>
<td>Transactions</td>
<td>r</td>
<td>Indexed access to all transaction components connected to this database.</td>
</tr>
</tbody>
</table>

### 11.19.4 TDatabase.Create

**Synopsis:** Initialize a new TDatabase class instance.

**Declaration:**

```pascal
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 (418), TDBTransaction (420), TDatabase.Destroy (366)

### 11.19.5 TDatabase.Destroy

**Synopsis:** Remove a TDatabase instance from memory.

**Declaration:**

```pascal
destructor Destroy; Override
```
CHAPTER 11. REFERENCE FOR UNIT 'DB'

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 (367)

11.19.6 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 (362), TDatabase.CloseTransactions (367)

11.19.7 TDatabase.CloseTransactions

Synopsis: End all transactions

Declaration: procedure CloseTransactions

Visibility: public

Description: CloseTransaction calls TDBTransaction.EndTransaction (420) on all connected transactions. It is called automatically when the connection is closed, after all datasets are closed.

See also: TCustomConnection.Close (362), TDatabase.CloseDatasets (367)

11.19.8 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 (420) component instead and invoke the TDBTransaction.StartTransaction (420) method.

See also: TDBTransaction (420), TDBTransaction.StartTransaction (420)

11.19.9 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 (420) component instead and invoke the TDBTransaction.EndTransaction (420) method.

See also: TDBTransaction (420), TDBTransaction.EndTransaction (420)
11.19.10 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 (368) array property.

**See also:** TDatabase.Transactions (368)

11.19.11 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 (368)

11.19.12 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 (365).

**See also:** TDatabase.Params (369), TDatabase.IsSQLBased (368)

11.19.13 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 (369), TDatabase.Directory (368)
11.19.14 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 (363).

See also: TCustomConnection.Connected (363)

11.19.15 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 (368) 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 (368), TDatabase.Params (369)

11.19.16 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 (369)

11.19.17 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 (368), TDatabase.DatabaseName (369)
11.20 TDataLink

11.20.1 Description
TDataLink is used by GUI controls or datasets in a master-detail relationship to handle data events coming from a TDatasource (412) instance. It is a class that exists for component programmers, application coders should never need to use TDataLink or one of its descendents.

DB-Aware Component coders must use a TDataLink instance to handle all communication with a TDataset (375) 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: TDatset (375), TDatasource (412), TDetailDatalink (424), TMasterDataLink (481)

11.20.2 Method overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>370</td>
<td>Create</td>
<td>Initialize a new instance of TDataLink</td>
</tr>
<tr>
<td>371</td>
<td>Destroy</td>
<td>Remove an instance of TDataLink from memory</td>
</tr>
<tr>
<td>371</td>
<td>Edit</td>
<td>Set the dataset in edit mode, if possible</td>
</tr>
<tr>
<td>371</td>
<td>ExecuteAction</td>
<td>Execute action</td>
</tr>
<tr>
<td>372</td>
<td>UpdateAction</td>
<td>Update handler for actions</td>
</tr>
<tr>
<td>371</td>
<td>UpdateRecord</td>
<td>Called when the data in the dataset must be updated</td>
</tr>
</tbody>
</table>

11.20.3 Property overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Properties</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>372</td>
<td>Active</td>
<td>r</td>
<td>Is the link active</td>
</tr>
<tr>
<td>372</td>
<td>ActiveRecord</td>
<td>rw</td>
<td>Currently active record</td>
</tr>
<tr>
<td>372</td>
<td>BOF</td>
<td>r</td>
<td>Is the dataset at the first record</td>
</tr>
<tr>
<td>373</td>
<td>BufferCount</td>
<td>rw</td>
<td>Set to the number of record buffers this datalink needs.</td>
</tr>
<tr>
<td>373</td>
<td>DataSet</td>
<td>r</td>
<td>Dataset this datalink is connected to</td>
</tr>
<tr>
<td>373</td>
<td>DataSource</td>
<td>rw</td>
<td>Datasource this datalink is connected to</td>
</tr>
<tr>
<td>373</td>
<td>DataSourceFixed</td>
<td>rw</td>
<td>Can the datasource be changed</td>
</tr>
<tr>
<td>374</td>
<td>Editing</td>
<td>r</td>
<td>Is the dataset in edit mode</td>
</tr>
<tr>
<td>374</td>
<td>Eof</td>
<td>r</td>
<td></td>
</tr>
<tr>
<td>374</td>
<td>ReadOnly</td>
<td>rw</td>
<td>Is the link readonly</td>
</tr>
<tr>
<td>374</td>
<td>RecordCount</td>
<td>r</td>
<td>Number of records in the buffer of the dataset</td>
</tr>
</tbody>
</table>

11.20.4 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 (371)
11.20.5  **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 (371)  

11.20.6  **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 (413) method of the DataSource (373) to which the datalink instance is coupled. If the TDatasource.AutoEdit (414) 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 (412), TDataLink.Datasource (373), TDatasource/Edit (413), TDatasource.AutoEdit (414)  

11.20.7  **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 (370) notification (which could result from writing data to the dataset) will be blocked.  
See also: TDataLink.RecordChange (370)  

11.20.8  **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 (372)
11.20.9  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 (371)

11.20.10  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 (403) whenever the DataSource (373) property is set.

See also: TDataLink.Datasource (373), TDataLink.ReadOnly (374), TDataset.Active (403)

11.20.11  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 (372), TDataLink.EOF (374)

11.20.12  TDataLink.BOF
Synopsis: Is the dataset at the first record

Declaration: Property BOF : Boolean

Visibility: public

Access: Read

Description: BOF returns TDataset.BOF (395) if the dataset is available, True otherwise.

See also: TDataLink.EOF (374), TDataset.BOF (395)
11.20.13 **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 (379), TDatalink.ActiveRecord (372)

11.20.14 **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 (373), TDataset (375)

11.20.15 **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` (412) instance to get access to the dataset it is connected to. A datalink never points directly to a `TDataset` (375) 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 (375), TDataSource (412)

11.20.16 **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` (373) 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 (373)
11.20.17  **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 (371) 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 (371), TDataset.Edit (384)

11.20.18  **TDataLink.Eof**

Synopsis:

Declaration: Property Eof : Boolean

Visibility: public

Access: Read

Description: EOF returns TDataset.EOF (397) if the dataset is available, True otherwise.

See also: TDataLink.BOF (372), TDataset.EOF (397)

11.20.19  **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 (371) 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 (372) property can be used to disable delivery of events to the datalink.

See also: TDataLink.Active (372), TDataLink.edit (371)

11.20.20  **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 (373) property: RecordCount is always less than Buffercount.

See also: TDataLink.BufferCount (373)
11.21 TDataSet

11.21.1 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. Therefore _TDataSet_ is never used directly, one always instantiates a descendent class.

Initially, no data is available: the dataset is inactive. The Open (392) 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 (403) property). Likewise, the Close (382) 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 (391), Prev (375), First (387) and Last (390) methods, it is possible to navigate through the records. The EOF (397) property will be _true_ if the last row has been reached. Likewise, the BOF (395) 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 (400) will be zero.

The Append (380) and Insert (389) methods can be used to insert new records to the set of records. The _TDataSet_.Delete (383) statement is used to delete the current record, and the _TDataSet_.Edit (384) 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 (392) method, or can be undone using the Cancel (381) method.

The data in the various fields properties is available through the _Fields_ (401) 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_ (424)
### 11.21.2 Method overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>379</td>
<td>ActiveBuffer</td>
<td>Currently active memory buffer</td>
</tr>
<tr>
<td>380</td>
<td>Append</td>
<td>Append a new record to the data</td>
</tr>
<tr>
<td>380</td>
<td>AppendRecord</td>
<td>Append a new record to the dataset and fill with data</td>
</tr>
<tr>
<td>381</td>
<td>BookmarkValid</td>
<td>Test whether ABookMark is a valid bookmark.</td>
</tr>
<tr>
<td>381</td>
<td>Cancel</td>
<td>Cancel the current editing operation</td>
</tr>
<tr>
<td>381</td>
<td>CheckBrowseMode</td>
<td>Check whether the dataset is in browse mode.</td>
</tr>
<tr>
<td>381</td>
<td>ClearFields</td>
<td>Clear the values of all fields</td>
</tr>
<tr>
<td>382</td>
<td>Close</td>
<td>Close the dataset</td>
</tr>
<tr>
<td>382</td>
<td>CompareBookmarks</td>
<td>Compare two bookmarks</td>
</tr>
<tr>
<td>382</td>
<td>ControlsDisabled</td>
<td>Check whether the controls are disabled</td>
</tr>
<tr>
<td>379</td>
<td>Create</td>
<td>Create a new TDataset instance</td>
</tr>
<tr>
<td>383</td>
<td>CreateBlobStream</td>
<td>Create blob stream</td>
</tr>
<tr>
<td>383</td>
<td>CursorPosChanged</td>
<td>Indicate a change in cursor position</td>
</tr>
<tr>
<td>383</td>
<td>DataConvert</td>
<td>Convert data from/to native format</td>
</tr>
<tr>
<td>383</td>
<td>Delete</td>
<td>Delete the current record.</td>
</tr>
<tr>
<td>379</td>
<td>Destroy</td>
<td>Free a TDataset instance</td>
</tr>
<tr>
<td>384</td>
<td>DisableControls</td>
<td>Disable event propagation of controls</td>
</tr>
<tr>
<td>384</td>
<td>Edit</td>
<td>Set the dataset in editing mode.</td>
</tr>
<tr>
<td>385</td>
<td>EnableControls</td>
<td>Enable event propagation of controls</td>
</tr>
<tr>
<td>385</td>
<td>FieldByName</td>
<td>Search a field by name</td>
</tr>
<tr>
<td>385</td>
<td>FindField</td>
<td>Find a field by name</td>
</tr>
<tr>
<td>386</td>
<td>FindFirst</td>
<td>Find the first active record (deprecated)</td>
</tr>
<tr>
<td>386</td>
<td>FindLast</td>
<td>Find the last active record (deprecated)</td>
</tr>
<tr>
<td>386</td>
<td>FindNext</td>
<td>Find the next active record (deprecated)</td>
</tr>
<tr>
<td>386</td>
<td>FindPrior</td>
<td>Find the previous active record (deprecated)</td>
</tr>
<tr>
<td>387</td>
<td>First</td>
<td>Position the dataset on the first record.</td>
</tr>
<tr>
<td>387</td>
<td>FreeBookmark</td>
<td>Free a bookmark obtained with GetBookmark (deprecated)</td>
</tr>
<tr>
<td>387</td>
<td>GetBookmark</td>
<td>Get a bookmark pointer (deprecated)</td>
</tr>
<tr>
<td>387</td>
<td>GetCurrentRecord</td>
<td>Copy the data for the current record in a memory buffer</td>
</tr>
<tr>
<td>379</td>
<td>GetFieldData</td>
<td>Get the data for a field</td>
</tr>
<tr>
<td>388</td>
<td>GetFieldList</td>
<td>Return field instances in a list</td>
</tr>
<tr>
<td>388</td>
<td>GetFieldNames</td>
<td>Return a list of all available field names</td>
</tr>
<tr>
<td>388</td>
<td>GotoBookmark</td>
<td>Jump to bookmark</td>
</tr>
<tr>
<td>389</td>
<td>Insert</td>
<td>Insert a new record at the current position.</td>
</tr>
<tr>
<td>389</td>
<td>InsertRecord</td>
<td>Insert a new record with given values.</td>
</tr>
<tr>
<td>389</td>
<td>IsEmpty</td>
<td>Check if the dataset contains no data</td>
</tr>
<tr>
<td>389</td>
<td>IsLinkedTo</td>
<td>Check whether a datasource is linked to the dataset</td>
</tr>
<tr>
<td>390</td>
<td>IsSequenced</td>
<td>Is the data sequenced</td>
</tr>
<tr>
<td>390</td>
<td>Last</td>
<td>Navigate forward to the last record</td>
</tr>
<tr>
<td>390</td>
<td>Locate</td>
<td>Locate a record based on some key values</td>
</tr>
<tr>
<td>391</td>
<td>Lookup</td>
<td>Search for a record and return matching values.</td>
</tr>
<tr>
<td>391</td>
<td>MoveBy</td>
<td>Move the cursor position</td>
</tr>
<tr>
<td>391</td>
<td>Next</td>
<td>Go to the next record in the dataset.</td>
</tr>
<tr>
<td>392</td>
<td>Open</td>
<td>Activate the dataset: Fetch data into memory.</td>
</tr>
<tr>
<td>392</td>
<td>Post</td>
<td>Post pending edits to the database.</td>
</tr>
<tr>
<td>393</td>
<td>Prior</td>
<td>Go to the previous record</td>
</tr>
<tr>
<td>393</td>
<td>Refresh</td>
<td>Refresh the records in the dataset</td>
</tr>
<tr>
<td>393</td>
<td>Resync</td>
<td>Resynchronize the data buffer</td>
</tr>
<tr>
<td>380</td>
<td>SetFieldData</td>
<td>Store the data for a field</td>
</tr>
<tr>
<td>393</td>
<td>SetFields</td>
<td>Set a number of field values at once</td>
</tr>
<tr>
<td>394</td>
<td>Translate</td>
<td>Transliterate a buffer</td>
</tr>
<tr>
<td>394</td>
<td>UpdateCursorPos</td>
<td>Update cursor position</td>
</tr>
<tr>
<td>394</td>
<td>UpdateRecord</td>
<td>Indicate that the record contents have changed</td>
</tr>
<tr>
<td>395</td>
<td>UpdateStatus</td>
<td>Get the update status for the current record</td>
</tr>
</tbody>
</table>
### Chapter 11. Reference for Unit 'DB'

#### 11.21.3 Property overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Properties</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>403</td>
<td>Active</td>
<td>rw</td>
<td>Is the dataset open or closed.</td>
</tr>
<tr>
<td>407</td>
<td>AfterCancel</td>
<td>rw</td>
<td>Event triggered after a Cancel operation.</td>
</tr>
<tr>
<td>404</td>
<td>AfterClose</td>
<td>rw</td>
<td>Event triggered after the dataset is closed.</td>
</tr>
<tr>
<td>407</td>
<td>AfterDelete</td>
<td>rw</td>
<td>Event triggered after a successful Delete operation.</td>
</tr>
<tr>
<td>405</td>
<td>AfterEdit</td>
<td>rw</td>
<td>Event triggered after the dataset is put in edit mode.</td>
</tr>
<tr>
<td>405</td>
<td>AfterInsert</td>
<td>rw</td>
<td>Event triggered after the dataset is put in insert mode.</td>
</tr>
<tr>
<td>404</td>
<td>AfterOpen</td>
<td>rw</td>
<td>Event triggered after the dataset is opened.</td>
</tr>
<tr>
<td>406</td>
<td>AfterPost</td>
<td>rw</td>
<td>Event called after changes have been posted to the underlying database.</td>
</tr>
<tr>
<td>408</td>
<td>AfterRefresh</td>
<td>rw</td>
<td>Event triggered after the data has been refreshed.</td>
</tr>
<tr>
<td>408</td>
<td>AfterScroll</td>
<td>rw</td>
<td>Event triggered after the cursor has changed position.</td>
</tr>
<tr>
<td>403</td>
<td>AutoCalcFields</td>
<td>rw</td>
<td>How often should the value of calculated fields be calculated.</td>
</tr>
<tr>
<td>406</td>
<td>BeforeCancel</td>
<td>rw</td>
<td>Event triggered before a Cancel operation.</td>
</tr>
<tr>
<td>404</td>
<td>BeforeClose</td>
<td>rw</td>
<td>Event triggered before the dataset is closed.</td>
</tr>
<tr>
<td>407</td>
<td>BeforeDelete</td>
<td>rw</td>
<td>Event triggered before a Delete operation.</td>
</tr>
<tr>
<td>405</td>
<td>BeforeEdit</td>
<td>rw</td>
<td>Event triggered before the dataset is put in edit mode.</td>
</tr>
<tr>
<td>404</td>
<td>BeforeInsert</td>
<td>rw</td>
<td>Event triggered before the dataset is put in insert mode.</td>
</tr>
<tr>
<td>403</td>
<td>BeforeOpen</td>
<td>rw</td>
<td>Event triggered before the dataset is opened.</td>
</tr>
<tr>
<td>406</td>
<td>BeforePost</td>
<td>rw</td>
<td>Event called before changes are posted to the underlying database.</td>
</tr>
<tr>
<td>408</td>
<td>BeforeRefresh</td>
<td>rw</td>
<td>Event triggered before the data is refreshed.</td>
</tr>
<tr>
<td>407</td>
<td>BeforeScroll</td>
<td>rw</td>
<td>Event triggered before the cursor changes position.</td>
</tr>
<tr>
<td>395</td>
<td>BlockReadSize</td>
<td>rw</td>
<td>Number of records to read</td>
</tr>
<tr>
<td>395</td>
<td>BOF</td>
<td>r</td>
<td>Is the cursor at the beginning of the data (on the first record)</td>
</tr>
<tr>
<td>395</td>
<td>Bookmark</td>
<td>rw</td>
<td>Get or set the current cursor position</td>
</tr>
<tr>
<td>396</td>
<td>CanModify</td>
<td>r</td>
<td>Can the data in the dataset be modified</td>
</tr>
<tr>
<td>397</td>
<td>DataSource</td>
<td>r</td>
<td>Datasource this dataset is connected to.</td>
</tr>
<tr>
<td>398</td>
<td>DefaultFields</td>
<td>r</td>
<td>Indicates whether the last record has been reached.</td>
</tr>
<tr>
<td>398</td>
<td>EOF</td>
<td>r</td>
<td>Is the dataset using persistent fields or not.</td>
</tr>
<tr>
<td>399</td>
<td>FieldCount</td>
<td>r</td>
<td>Number of fields</td>
</tr>
<tr>
<td>399</td>
<td>FieldDefs</td>
<td>rw</td>
<td>Definitions of available fields in the underlying database</td>
</tr>
<tr>
<td>401</td>
<td>Fields</td>
<td>r</td>
<td>Indexed access to the fields of the dataset.</td>
</tr>
<tr>
<td>401</td>
<td>FieldValues</td>
<td>rw</td>
<td>Access to field values based on the field names.</td>
</tr>
<tr>
<td>402</td>
<td>Filter</td>
<td>rw</td>
<td>Filter to apply to the data in memory.</td>
</tr>
<tr>
<td>402</td>
<td>Filtered</td>
<td>rw</td>
<td>Is the filter active or not.</td>
</tr>
<tr>
<td>402</td>
<td>FilterOptions</td>
<td>rw</td>
<td>Options to apply when filtering</td>
</tr>
<tr>
<td>399</td>
<td>Found</td>
<td>r</td>
<td>Check success of one of the Find methods</td>
</tr>
<tr>
<td>399</td>
<td>IsUniDirectional</td>
<td>r</td>
<td>Is the dataset unidirectional (i.e. forward scrolling only)</td>
</tr>
<tr>
<td>399</td>
<td>Modified</td>
<td>r</td>
<td>Was the current record modified?</td>
</tr>
<tr>
<td>409</td>
<td>OnCalcFields</td>
<td>rw</td>
<td>Event triggered when values for calculated fields must be computed.</td>
</tr>
<tr>
<td>409</td>
<td>OnDeleteError</td>
<td>rw</td>
<td>Event triggered when a delete operation fails.</td>
</tr>
<tr>
<td>409</td>
<td>OnEditError</td>
<td>rw</td>
<td>Event triggered when an edit operation fails.</td>
</tr>
<tr>
<td>410</td>
<td>OnFilterRecord</td>
<td>rw</td>
<td>Event triggered to filter records.</td>
</tr>
<tr>
<td>410</td>
<td>OnNewRecord</td>
<td>rw</td>
<td>Event triggered when a new record is created.</td>
</tr>
<tr>
<td>411</td>
<td>OnPostError</td>
<td>rw</td>
<td>Event triggered when a post operation fails.</td>
</tr>
<tr>
<td>400</td>
<td>RecNo</td>
<td>rw</td>
<td>Current record number</td>
</tr>
<tr>
<td>400</td>
<td>RecordCount</td>
<td>r</td>
<td>Number of records in the dataset</td>
</tr>
<tr>
<td>400</td>
<td>RecordSize</td>
<td>r</td>
<td>Size of the record in memory</td>
</tr>
<tr>
<td>401</td>
<td>State</td>
<td>r</td>
<td>Current operational state of the dataset</td>
</tr>
</tbody>
</table>
11.21.4 TDataSet.Create

Synopsis: Create a new TDataSet instance

Declaration: constructor Create(AOwner: TComponent); Override

Visibility: public

Description: Create initializes a new TDataSet 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 (379)

11.21.5 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 the 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 (382), TDataSet.Create (379)

11.21.6 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.

11.21.7 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 descendant 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).
CHAPTER 11. REFERENCE FOR UNIT ‘DB’

11.21.8 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 (438)

11.21.9 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 (389) 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 (389), TDataset.Edit (384)

11.21.10 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 (393)) and attempts to post the record using TDataset.Post (392). 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 (380), TDataset.SetFields (393), TDataset.Post (392)
11.21.11  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 (387), TDataset.SetBookmark (375), TDataset.FreeBookmark (387), TDataset.BookmarkAvailable (375)

11.21.12  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 (406) and TDataset.AfterCancel (407) events. If the dataset was in insert mode, then the TDataset.BeforeScroll (407) and TDataset.AfterScroll (408) 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 (401), TDataset.Append (380), TDataset.Insert (389), TDataset.Edit (384)

11.21.13  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 (338) exception is raised. If it is active, but in an edit state, then TDataset.UpdateRecord (394) is called, and if the TDataset.Modified (399) property is true, a TDataset.Post (392) is performed, else TDataset.Cancel (381) is called.

See also: TDataset.State (401), TDataset.Post (392), TDataset.Cancel (381), TDataset.UpdateRecord (394), TDataset.Modified (399)

11.21.14  TDataSet.ClearFields

Synopsis: Clear the values of all fields

Declaration: procedure ClearFields

Visibility: public
CHAPTER 11. REFERENCE FOR UNIT 'DB'

Description: ClearFields clears the values of all fields.

Errors: If the dataset is not in editing mode (State in dsEditmodes), then an EDatabaseError (338) exception will be raised.

See also: TDataSet.State (401), TField.Clear (429)

11.21.15 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 (404) and TDataSet.AfterClose (404) 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 (403), TDataSet.Open (392)

11.21.16 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 (384) call can be used to disable sending of data events to the controls. The sending can be re-enabled with TDataSet.EnableControls (385). 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 (384), TDataSet.EnableControls (385)

11.21.17 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 descendant 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 (381), TDataSet.GetBookmark (387), TDataSet.SetBookmark (375)
### 11.21.18 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` (353) and `TBlobField.SaveToStream` (353) 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` (353), `TBlobField.SaveToStream` (353)

### 11.21.19 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).

### 11.21.20 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` (379), `TDataset.SetFieldData` (380)

### 11.21.21 TDataSet.Delete

**Synopsis:** Delete the current record.

**Declaration:**
```
procedure Delete; Virtual
```

---

383
Visibility: public

Description: Delete will delete the current record. This action will trigger the TDataset.BeforeDelete (407), TDataset.BeforeScroll (407), TDataset.AfterDelete (407) and TDataset.AfterScroll (408) 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 (338) exception will be raised.

See also: TDataset.Cancel (381), TDataset.BeforeDelete (407), TDataset.BeforeScroll (407), TDataset.AfterDelete (407), TDataset.AfterScroll (408)

11.21.22 TDataSetDisableControls

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 (385). 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:

```delphi
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 (382).

See also: TDataset.EnableControls (385), TDataset.ControlsDisabled (382)

11.21.23 TDataSetEdit

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 (405) and TDataset.AfterEdit (405) 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 (380) instead.

Errors: If the dataset is read-only or not opened, then an EDatabaseError (338) exception will be raised.

See also: TDataset.State (401), TDataset.EOF (397), TDataset.BOF (395), TDataset.Append (380), TDataset.BeforeEdit (405), TDataset.AfterEdit (405)
11.21.24 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 (384) 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 (382).

See also: TDataSet.DisableControls (384), TDataSet.ControlsDisabled (382)

11.21.25 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 (460): it searches for the field with fieldname equaling FieldName. The case is performed case-insensitive. The matching field instance is returned.

Errors: If the field is not found, an EDatabaseError (338) exception will be raised.

See also: TFields.FieldByName (460), TDataSet.FindField (385)

11.21.26 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 (460): it searches for the field with fieldname equaling 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 (385), TFields.FindField (460)
11.21.27  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 (387) instead.

See also: TDataSet.First (387), TDataSet.FindLast (386), TDataSet.FindNext (386), TDataSet.FindPrior (386)

11.21.28  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 (390) instead.

See also: TDataSet.Last (390), TDataSet.FindFirst (386), TDataSet.FindNext (386), TDataSet.FindPrior (386)

11.21.29  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 (391) instead.

See also: TDataSet.Next (391), TDataSet.FindFirst (386), TDataSet.FindLast (386), TDataSet.FindPrior (386)

11.21.30  TDataSet.FindPrior

Synopsis: Find the previous active record (deprecated)

Declaration: function FindPrior : Boolean; Virtual

Visibility: public
CHAPTER 11. REFERENCE FOR UNIT 'DB'

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 (393) instead.

See also: TDataset.Prior (393), TDataset.FindFirst (386), TDataset.FindLast (386), TDataset.FindPrior (386)

### 11.21.31 TDataSet.First

**Synopsis:** Position the dataset on the first record.

**Declaration:**

```cpp
procedure First
```

**Visibility:** public

**Description:** **First** positions the dataset on the first record. This action will trigger the TDataset.BeforeScroll (407) and TDataset.AfterScroll (408) events. After the action is completed, the TDataset.BOF (395) property will be **True**.

**Errors:** If the dataset is unidirectional or is closed, an EDatabaseError (338) exception will be raised.

See also: TDataset.Prior (393), TDataset.Last (390), TDataset.Next (391), TDataset.BOF (395), TDataset.BeforeScroll (407), TDataset.AfterScroll (408)

### 11.21.32 TDataSet.FreeBookmark

**Synopsis:** Free a bookmark obtained with GetBookmark (deprecated)

**Declaration:**

```cpp
procedure FreeBookmark(ABookmark: TBookmark); Virtual
```

**Visibility:** public

**Description:** **FreeBookmark** must be used to free a bookmark obtained by TDataset.GetBookmark (387). It should not be used on bookmarks obtained with the TDataset.Bookmark (395) 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 (387), TDataset.Bookmark (395)

### 11.21.33 TDataSet.GetBookmark

**Synopsis:** Get a bookmark pointer (deprecated)

**Declaration:**

```cpp
function GetBookmark : TBookmark; Virtual
```

**Visibility:** public

**Description:** **GetBookmark** gets a bookmark pointer to the current cursor location. The TDataset.SetBookmark (375) 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 (387) call. The bookmark will be **Nil** if the dataset is empty or not active.

This call is deprecated. Use the TDataset.Bookmark (395) property instead to get a bookmark.

See also: TDataset.SetBookmark (375), TDataset.FreeBookmark (387), TDataset.Bookmark (395)
**11.21.34 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 (379)

**11.21.35 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 (338) exception will be raised.

See also: TDataset.GetFieldNames (388), TDataset.FieldByName (385), TDataset.FindField (385)

**11.21.36 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 (375), TDataset.FieldByName (385), TDataset.FindField (385)

**11.21.37 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 (387) function.

This function is deprecated, use the TDataset.Bookmark (395) property instead.

Errors: if ABookmark does not contain a valid bookmark, then an exception may be raised.

See also: TDataset.Bookmark (395), TDataset.GetBookmark (387), TDataset.FreeBookmark (387)
11.21.38  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 (404), BeforeScroll (407), OnNewRecord (410), AfterInsert (405) and AfterScroll (408) 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 (392) must be called.

Errors: If the dataset is read-only, calling Insert will result in an EDatabaseError (338).

See also: BeforeInsert (404), BeforeScroll (407), OnNewRecord (410), AfterInsert (405), AfterScroll (408), TDataSet.Post (392), TDataSet.Append (380)

11.21.39  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 (389), TDataSet.SetFieldValues (375)

11.21.40  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 (397) and TDataSet.BOF (395) are both True, and the dataset is not in insert mode.

See also: TDataSet.EOF (397), TDataSet.BOF (395), TDataSet.State (401)

11.21.41  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 (415) points to the current dataset instance, or indirectly.

See also: TDataSource.Dataset (415)
11.21.42 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 (400) 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 (400)

11.21.43 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 (391). After a call to Last, the TDataSet.EOF (397) property will be True. Calling this method will trigger the TDataSet.BeforeScroll (407) and TDataSet.AfterScroll (408) events.

See also: TDataSet.First (387), TDataSet.Next (391), TDataSet.EOF (397), TDataSet.BeforeScroll (407), TDataSet.AfterScroll (408)

11.21.44 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 (330).

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 (375), TDataSet.Lookup (391), TLocateOption (330)
11.21.45 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 (390) 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 (338) exception will be raised.

See also: TDataset.Locate (390)

11.21.46 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 (407) and TDataset.AfterScroll (408) 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 (338) exception on unidirectional datasets.

See also: TDataset.RecNo (400), TDataset.BeforeScroll (407), TDataset.AfterScroll (408)

11.21.47 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 (407) and TDataset.AfterScroll (408) events. If the dataset is located on the last known record (EOF (397) 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 (338) exception.

See also: TDataset.MoveBy (391), TDataset.Prior (393), TDataset.Last (390), TDataset.BeforeScroll (407), TDataset.AfterScroll (408), TDataset.EOF (397)
11.21.48 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 (395) is set to True
4. Internal buffers are allocated and filled with data
5. If the dataset is empty, EOF (397) is set to true
6. State (401) is set to dsBrowse
7. The AfterOpen (404) 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 (403), TDataset.State (401), TDataset.BOF (395), TDataset.EOF (397), TDataset.BeforeOpen (403), TDataset.AfterOpen (404)

11.21.49 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 (389), Append (380) or TDataSet.Edit (384) 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 (406), AfterPost (406) events. After the call to Post, the State (401) 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 (320)) will result in an EdatabaseError (338) exception. If an exception occurs during the save operation, the OnPostError (411) event is triggered to handle the error.

See also: Insert (389), Append (380), Edit (384), OnPostError (411), BeforePost (406), AfterPost (406), State (401)
11.21.50  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 (407) and TDataSet.AfterScroll (408) events. If the dataset is located on the first record, (BOF (395) 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 (338) exception.

See also: TDataSet.MoveBy (391), TDataSet.Next (391), TDataSet.First (387), TDataSet.BeforeScroll (407), TDataSet.AfterScroll (408), TDataSet.BOF (395)

11.21.51  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 (408) and AfterRefresh (408) events.

Errors: Refreshing may fail if the underlying dataset descendent does not support it.

See also: TDataSet.Close (382), TDataSet.Open (392), BeforeRefresh (408), AfterRefresh (408)

11.21.52  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 (393) should be used. The Resync parameter indicates how the buffers should be refreshed.

See also: TDataSet.Refresh (393)

11.21.53  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 (401) property, and works its way down the array.

Errors: If the dataset is not in edit mode, then an EDatabaseError (338) exception will be raised. If there are more values than fields, an EListError exception will be raised.

See also: TDataset.Fields (401)

### 11.21.54 TDataSet.Translate

Synopsis: Transliterate a buffer

Declaration: function Translate(Src: PChar; Dest: PChar; ToOem: Boolean) : Integer

Visibility: public

Description: Translate is called for all string fields for which the TStringField.Transliterate (510) 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 descendants 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 buffers.

See also: TStringField.Transliterate (510)

### 11.21.55 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 (393)

### 11.21.56 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 descendants of TDataset.

See also: OnUpdateRecord (375)
11.21.57 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.

11.21.58 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 (319), DisableControls (319)

11.21.59 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 (387). As soon as TDataset.Next (391) is called, BOF will no longer be true.

See also: TDataset.EOF (397), TDataset.Next (391), TDataset.First (387)

11.21.60 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:

```pascal
Var
  B : TBookmarkStr;
begin
  With MyDataset do
    begin
      B:=Bookmark;
      DisableControls;
      try
        First;
        While Not EOF do
          begin
            DoSomething;
            Next;
          end;
      finally
        EnableControls;
        Bookmark:=B;
      end;
    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 (384) and TDataset.EnableControls (385) 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 (384), TDataset.EnableControls (385)

### 11.21.61 TDataSet.CanModify

**Synopsis:** Can the data in the dataset be modified

**Declaration:**

```pascal
Property CanModify : Boolean
```

**Visibility:** public

**Access:** Read

**Description:** CanModify 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 (389), TDataset.Append (380), TDataset.Delete (383), Tdataset.Edit (384)
11.21.62  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 (412)

11.21.63  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 (398) 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 (398), TDataset.Fields (401), TField (424)

11.21.64  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 (391) method is called.
2. The TDataset.Last (390) method is called (which is equivalent to moving to the last record and calling TDataset.Next (391)).
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:

```delphi
With MyDataset do
  While not EOF do
    begin
      DoSomething;
      Next;
    end;
```

See also: `TDataset.BOF` (395), `TDataset.Next` (391), `TDataset.Last` (390), `TDataset.IsEmpty` (389)

### 11.21.65 TDataSet.FieldCount

**Synopsis:** Number of fields

**Declaration:**
```
Property FieldCount : LongInt
```

**Visibility:** public  
**Access:** Read

**Description:** FieldCount is the same as `Fields.Count` (462), i.e. the number of fields. For a dataset with persistent fields (when `DefaultFields` (397) 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` (458)

### 11.21.66 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 (397) 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` (392), `TDataset.DefaultFields` (397), `TDataset.Fields` (401)
**11.21.67  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 (386), TDataSet.FindLast (386), TDataSet.FindNext (386) or TDataSet.FindPrior (386) operations was successful.

**See also:** TDataSet.FindFirst (386), TDataSet.FindLast (386), TDataSet.FindNext (386), TDataSet.FindPrior (386)

---

**11.21.68  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 (384) or Tdataset.Insert (389). It becomes True if a value was written to one of the fields of the dataset.

**See also:** Tdataset.Edit (384), TDataset.Insert (389), TDataset.Append (380), TDataset.Cancel (381), TDataset.Post (392)

---

**11.21.69  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 (393) or TDataset.Last (390)), random positioning of the cursor, editing or filtering will result in an EDatabaseError (338). 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 (393), TDataset.Next (391)
11.21.70  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 descendant 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 therefore 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 (400)

11.21.71  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 (400)

11.21.72  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 descendant manages it's data.

See also: TField.Datasize (438), TDataset.RecordCount (400), TDataset.RecNo (400)
11.21.73  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 (403), TDataSet.Edit (384), TDataSet.Insert (389), TDataSet.Append (380), TDataSet.Post (392), TDataSet.Cancel (381)

11.21.74  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 (458) 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 (455), TField (424)

11.21.75  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 (460)

401
11.21.76  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 (402), TDataset.FilterOptions (402)

11.21.77  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 (402) 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 (402), TDataset.FilterOptions (402)

11.21.78  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 (402), 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 (402), TDataset.FilterOptions (402)
11.21.79 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 (401) is then dsInactive). Setting the Active property to True is equivalent to calling TDataSet.Open (392), setting it to False is equivalent to calling TDataSet.Close (382).

See also: TDataSet.State (401), TDataSet.Open (392), TDataSet.Close (382)

11.21.80 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 (409) event.

See also: TField.FieldKind (445), TDataSet.OnCalcFields (409)

11.21.81 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 (404), TDataSet.State (401)
11.21.82 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 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 (404), TDataSet.State (401), TDataSet.AfterScroll (408)

11.21.83 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 (404), TDataSet.State (401)

11.21.84 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 (404), TDataSet.State (401)

11.21.85 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 (380) or TDataset.Insert (389) 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 (405), TDataset.Append (380), TDataset.Insert (389)

11.21.86 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 (408) 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 (404), TDataset.AfterScroll (408), TDataset.Append (380), TDataset.Insert (389)

11.21.87 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 triggered at the start of the TDataset.Edit (384) 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 (405), TDataset.Edit (384), TDataset.State (401)

11.21.88 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 (405), TDataset.Edit (384), TDataset.State (401)
11.21.89  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 (392) 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 (392), TDataSet.AfterPost (406), TDataSet.State (401)

11.21.90  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 (392) 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.OnPostExecute (411) event is triggered instead.

See also: TDataSet.BeforePost (406), TDataSet.Post (392), TDataSet.State (401), TDataSet.OnPostExecute (411)

11.21.91  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 (381) 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 (407), TDataSet.Cancel (381), TDataSet.State (401)
11.21.92  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 (381) operation was successfully completed, and the dataset is again in dsBrowse state.

See also: TDataSet_BeforeCancel (406), TDataSet_Cancel (381), TDataSet_State (401)

11.21.93  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 (383) 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 (407) event. If the dataset was in insert mode when the Delete method was called, then the event will not be called, as TDataSet_Cancel (381) is called instead.

See also: TDataSet_AfterDelete (407), TDataSet_Delete (383), TDataSet_BeforeScroll (407), TDataSet_Cancel (381), TDataSet_State (401)

11.21.94  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 (383) operation, when the dataset is again in dsBrowse state. The event is followed by a TDataSet_AfterScroll (408) event.

See also: TDataSet_BeforeDelete (407), TDataSet_Delete (383), TDataSet_AfterScroll (408), TDataSet_State (401)

11.21.95  TDataSet.BeforeScroll
Synopsis: Event triggered before the cursor changes position.

Declaration: Property BeforeScroll : TDataSetNotifyEvent
Visibility: public
CHAPTER 11. REFERENCE FOR UNIT 'DB'

Access: Read, Write

Description: BeforeScroll is triggered before the cursor changes position. This can happen with one of the navigation methods: TDataSet.Next (391), TDataSet.Prior (393), TDataSet.First (387), TDataSet.Last (390), but also with two of the editing operations: TDataSet.Insert (389) and TDataSet.Delete (383). Raising an exception in this event handler aborts the operation in progress.

See also: TDataSet.AfterScroll (408), TDataSet.Next (391), TDataSet.Prior (393), TDataSet.First (387), TDataSet.Last (390), TDataSet.Insert (389), TDataSet.Delete (383)

11.21.96 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 (391), TDataSet.Prior (393), TDataSet.First (387), TDataSet.Last (390), but also with two of the editing operations: TDataSet.Insert (389) and TDataSet.Delete (383) 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 (408), TDataSet.Next (391), TDataSet.Prior (393), TDataSet.First (387), TDataSet.Last (390), TDataSet.Insert (389), TDataSet.Delete (383), TDataSet.Open (392)

11.21.97 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 (393) 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 (393), TDataSet.AfterRefresh (408), TDataSet.State (401)

11.21.98 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 (393) 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 (393), TDataSet.State (401), TDataSet.BeforeRefresh (408)
11.21.99  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 (403) 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 (403), TField.Kind (424)

11.21.100  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 (383) 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:

- daFail: The operation should fail (an exception will be raised)
- daAbort: The operation should be aborted (edits are undone, and an EAbort exception is raised)
- daRetry: Retry the operation.

For more information, see also the description of the TDataSetErrorEvent (324) event handler type.

See also: TDataSetErrorEvent (324), TDataSet.Delete (383), TDataSet.OnEditError (409), TDataSet.OnPostError (411)

11.21.101  TDataSet.OnEditError

Synopsis: Event triggered when an edit operation fails.

Declaration: Property OnEditError : TDataSetErrorEvent

Visibility: public
Access: Read, Write
CHAPTER 11. REFERENCE FOR UNIT 'DB'

**Description:** `OnEditError` is triggered when the `TDataset.Edit` 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:

- **daFail**: The operation should fail (an exception will be raised)
- **daAbort**: The operation should be aborted (edits are undone, and an `EAbort` exception is raised)
- **daRetry**: Retry the operation.

For more information, see also the description of the `TDatasetErrorEvent` event handler type.

**See also:** `TDatasetErrorEvent`, `TDataset.Edit`, `TDataset.OnDeleteError`, `TDataset.OnPostError`

---

**11.21.102 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` 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`, `TDataset.Filtered`, `TDataset.state`

---

**11.21.103 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` or `TDataset.Insert` 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` is again `dsBrowse`).

**See also:** `TDataset.Append`, `TDataset.Insert`, `TDataset.State`
11.21.104  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 (392) 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:

- daFail: The operation should fail (an exception will be raised)
- daAbort: The operation should be aborted (edits are undone, and an EAbort exception is raised)
- daRetry: Retry the operation.

For more information, see also the description of the TDataSetErrorEvent (324) event handler type.

See also: TDataSetErrorEvent (324), TDataSet.Post (392), TDataSet.OnDeleteError (409), TDataSet.OnEditError (409)

11.22  TDataSetEnumerator

11.22.1  Description

TDataSetEnumerator is an enumerator for the records in a dataset. It returns the TDataSet.Fields (401) 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:

```pascal
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 (375), TFields (458), TFieldsEnumerator (462)

11.22.2  Method overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>412</td>
<td>Create</td>
<td>Create a new instance of the dataset enumerator</td>
</tr>
<tr>
<td>412</td>
<td>MoveNext</td>
<td>Attempts to navigate to the next record</td>
</tr>
</tbody>
</table>
11.22.3 Property overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Properties</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>412</td>
<td>Current</td>
<td>r</td>
<td>Current record</td>
</tr>
</tbody>
</table>

11.22.4 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 first record.

Errors: None.

See also: TDataset (375), TDataset.First (387), TFieldsEnumerator (462)

11.22.5 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 (391), TDataset.EOF (397)

11.22.6 TDataSetEnumerator.Current

Synopsis: Current record

Declaration: Property Current : TFields

Visibility: public

Access: Read

Description: Current always returns TDataset.Fields (401).

See also: TDataset.Fields (401)

11.23 TDataSource

11.23.1 Description

TDataSource is a mediating component: it handles communication between any DB-Aware component (often edit controls on a form) and a TDataset (375) instance. Any database aware component should never communicate with a dataset directly. Instead, it should communicate with a TDataSource (412) 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 (375), TDataLink (370)

### 11.23.2 Method overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>413</td>
<td>Create</td>
<td>Create a new instance of TDatasource</td>
</tr>
<tr>
<td>413</td>
<td>Destroy</td>
<td>Remove a TDatasource instance from memory</td>
</tr>
<tr>
<td>413</td>
<td>Edit</td>
<td>Put the dataset in edit mode, if needed</td>
</tr>
<tr>
<td>414</td>
<td>IsLinkedTo</td>
<td>Check if a dataset is linked to a certain dataset</td>
</tr>
</tbody>
</table>

### 11.23.3 Property overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Properties</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>414</td>
<td>AutoEdit</td>
<td>rw</td>
<td>Should the dataset be put in edit mode automatically</td>
</tr>
<tr>
<td>415</td>
<td>DataSet</td>
<td>rw</td>
<td>Dataset this datasource is connected to</td>
</tr>
<tr>
<td>415</td>
<td>Enabled</td>
<td>rw</td>
<td>Enable or disable sending of events</td>
</tr>
<tr>
<td>416</td>
<td>OnDataChange</td>
<td>rw</td>
<td>Called whenever data changes in the current record</td>
</tr>
<tr>
<td>415</td>
<td>OnStateChange</td>
<td>rw</td>
<td>Called whenever the state of the dataset changes</td>
</tr>
<tr>
<td>416</td>
<td>OnUpdateData</td>
<td>rw</td>
<td>Called whenever the data in the dataset must be updated</td>
</tr>
<tr>
<td>414</td>
<td>State</td>
<td>r</td>
<td>State of the dataset</td>
</tr>
</tbody>
</table>

### 11.23.4 TDataSource.Create

**Synopsis:** Create a new instance of TDataSource

**Declaration:**
```pascal
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 (413)

### 11.23.5 TDataSource.Destroy

**Synopsis:** Remove a TDataSource instance from memory

**Declaration:**
```pascal
destructor Destroy; Override
```

**Visibility:** public

**Description:** Destroy notifies all TDataLink (370) instances connected to it that the dataset is no longer available, and then removes itself from the TDataLink instance. It then cleans up all resources and calls the inherited constructor.

**See also:** TDataSource.Create (413), TDataLink (370)

### 11.23.6 TDataSource.Edit

**Synopsis:** Put the dataset in edit mode, if needed

**Declaration:**
```pascal
procedure Edit
```

**Visibility:** public
Description: Edit will check AutoEdit (414); if it is True, then it puts the Dataset (415) 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 (384) directly.

Errors: An EDatabaseError (338) 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 (414), TDataSet.Edit (384), TDataSet.State (401)

11.23.7 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 (415) 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 (415)

11.23.8 TDataSource.State

Synopsis: State of the dataset

Declaration: Property State : TDataSetState

Visibility: public
Access: Read

Description: State contains the State (401) 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 (401) directly, to take into account the effect of the Enabled (415) property.

See also: TDataSet.State (401), TDataSource.Enabled (415)

11.23.9 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 (413) 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 (413), TDataSet.Edit (384)
11.23.10  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 (375) 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 (375), TDataSource.Enabled (415)

11.23.11  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 (375), TDataSource.Dataset (415), TDataSource.AutoEdit (414)

11.23.12  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 (401) 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 (416), TDataSource.OnStateChange (415), TDataset.State (401), TDataSource.Enabled (415)
11.23.13  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 (416)`, `TDataSource.OnStateChange (415)`, `TDataset.AfterScroll (408)`, `TFieldOnChange (449)`, `TDataSource.Enabled (415)`

11.23.14  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 (392)` 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 (416)`, `TDataSource.OnStateChange (415)`, `TDataset.Post (392)`

11.24  TDateField

11.24.1  Description

`TDateField` is the class used when a dataset must manage data of type date. (TField.DataType (438) equals `ftDate`). It initializes some of the properties of the TField (424) 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 (375)`, `TField (424)`, `TDateTimeField (417)`, `TTimeField (511)`

11.24.2  Method overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>417</td>
<td>Create</td>
<td>Create a new instance of a <code>TDateField</code> class.</td>
</tr>
</tbody>
</table>
11.24.3 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 (424) properties to configure the instance for working with date values.

See also: TField (424)

11.25 TDateTimeField

11.25.1 Description

TDateTimeField is the class used when a dataset must manage data of type datetime. (TField.DataType (438) equals ftDateTime). It also serves as base class for the TDateField (416) or TTimeField (511) classes. It overrides some of the properties and methods of the TField (424) 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 (375), TField (424), TDateField (416), TTimeField (511)

11.25.2 Method overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>417</td>
<td>Create</td>
<td>Create a new instance of a TDateTimeField class.</td>
</tr>
</tbody>
</table>

11.25.3 Property overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Properties</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>418</td>
<td>DisplayFormat</td>
<td>rw</td>
<td>Formatting string for textual representation of the field</td>
</tr>
<tr>
<td>418</td>
<td>EditMask</td>
<td></td>
<td>Specify an edit mask for an edit control</td>
</tr>
<tr>
<td>418</td>
<td>Value</td>
<td>rw</td>
<td>Contents of the field as a TDateTime value</td>
</tr>
</tbody>
</table>

11.25.4 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 (424) properties to configure the instance for working with date/time values.

See also: TField (424)
11.25.5  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 (442) by TDateTimeField as a TDateTime value. It returns the same value as the TField.AsDateTime (433) property.

See also: TField.AsDateTime (433), TField.Value (442)

11.25.6  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 (438) property to format the value with the DateTimeToString (??) function.

See also: DateTimeToString (??), FormatDateTime (??), TField.DisplayText (438)

11.25.7  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/EditMark (439)

11.26  TDBDataset

11.26.1  Description
TDBDataset is a TDataset descendent which introduces the concept of a database: a central component (TDatabase (365)) which represents a connection to a database. This central component is exposed in the TDBDataset.Database (419) 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 (420) 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 (365), TDBTransaction (420)

### 11.26.2 Method overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>419</td>
<td>destroy</td>
<td>Remove the TDBDataset instance from memory.</td>
</tr>
</tbody>
</table>

### 11.26.3 Property overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Properties</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>419</td>
<td>DataBase</td>
<td>rw</td>
<td>Database this dataset is connected to</td>
</tr>
<tr>
<td>420</td>
<td>Transaction</td>
<td>rw</td>
<td>Transaction in which this dataset is running.</td>
</tr>
</tbody>
</table>

### 11.26.4 TDBDataset.destroy

**Synopsis:** Remove the TDBDataset instance from memory.

**Declaration:**

destructor destroy;  Override

**Visibility:** public

**Description:** Destroy will disconnect the TDBDataset from its Database (419) and Transaction (420). After this it calls the inherited destructor.

See also: TDBDataset.Database (419), TDatabase (365)

### 11.26.5 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 (365) 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 (338) exception will be raised.

See also: TDatabase (365), TDBDataset.Transaction (420)
11.26.6 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 (420) 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 (420), TDBDataset.Database (419)

11.27 TDBTransaction

11.27.1 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 (365) instance, and to keep track of TDataset instances which are connected to the transaction.

See also: TDatabase (365), TDataset (375)

11.27.2 Method overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>421</td>
<td>CloseDataSets</td>
<td>Close all connected datasets</td>
</tr>
<tr>
<td>420</td>
<td>Create</td>
<td>Transaction property</td>
</tr>
<tr>
<td>421</td>
<td>Destroy</td>
<td>Remove a TDBTransaction instance from memory.</td>
</tr>
</tbody>
</table>

11.27.3 Property overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Properties</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>421</td>
<td>Active</td>
<td>rw</td>
<td>Is the transaction active or not</td>
</tr>
<tr>
<td>421</td>
<td>DataBase</td>
<td>rw</td>
<td>Database this transaction is connected to</td>
</tr>
</tbody>
</table>

11.27.4 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 (421)
11.27.5 TDBTransaction.Destroy

Synopsis: Remove a TDBTransaction instance from memory.

Declaration: destructor Destroy; Override

Visibility: public

Description: Destroy first disconnects all connected TDBDataset (418) instances and then cleans up the resources allocated in the Create (420) constructor. After that it calls the inherited destructor.

See also: TDBTransaction.Create (420)

11.27.6 TDBTransaction.CloseDataSets

Synopsis: Close all connected datasets

Declaration: procedure CloseDataSets

Visibility: public

Description: CloseDataSets closes all connected datasets (All TDBDataset (418) instances whose Transaction (420) property points to this TDBTransaction instance).

See also: TDBDataset (418), TDBDataset.Transaction (420)

11.27.7 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 (338) exception.

See also: TDBTransaction.Active (421), TDatabase (365)

11.27.8 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 (420). Reversely, setting Active to True will call StartTransaction, setting it to False will call TDBTransaction.EndTransaction (420).

See also: TDBTransaction.StartTransaction (420), TDBTransaction.EndTransaction (420)
11.28  TDefCollection

11.28.1  Description

TDefCollection is a parent class for the TFieldDefs (455) and TIndexDefs (473) collections: It holds a set of named definitions on behalf of a TDataset (375) component. To this end, it introduces a dataset (423) property, and a mechanism to notify the dataset of any updates in the collection. It is supposed to hold items of class TNamedItem (486), so the TDefCollection.Find (422) method can find items by named.

11.28.2  Method overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>422</td>
<td>create</td>
<td>Instantiate a new TDefCollection instance.</td>
</tr>
<tr>
<td>422</td>
<td>Find</td>
<td>Find an item by name</td>
</tr>
<tr>
<td>423</td>
<td>GetItemNames</td>
<td>Return a list of all names in the collection</td>
</tr>
<tr>
<td>423</td>
<td>IndexOf</td>
<td>Find location of item by name</td>
</tr>
</tbody>
</table>

11.28.3  Property overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Properties</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>423</td>
<td>Dataset</td>
<td>r</td>
<td>Dataset this collection manages definitions for.</td>
</tr>
<tr>
<td>423</td>
<td>Updated</td>
<td>rw</td>
<td>Has one of the items been changed</td>
</tr>
</tbody>
</table>

11.28.4  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 (375), TNamedItem (486)

11.28.5  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 (487), TDefCollection.IndexOf (423)
11.28.6 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 (487)

11.28.7 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 (422), TNamedItem.Name (487)

11.28.8 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.

11.28.9 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.
11.29 TDetailDataLink

11.29.1 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 (375), TDatasource (412)

11.29.2 Property overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Properties</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>424</td>
<td>DetailDataSet</td>
<td>r</td>
<td>Detail dataset in Master-detail relation</td>
</tr>
</tbody>
</table>

11.29.3 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 (481). The master dataset is available through the regular TDataLink.DataSource (373) property.

See also: TDataset (375), TMasterDatalink (481), TDataLink.DataSource (373)

11.30 TField

11.30.1 Description

TField is an abstract class that defines access methods for a field in a record, controlled by a TDataset (375) 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 (375), TFieldDef (450), TFields (458)
### 11.30.2 Method overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>428</td>
<td>Assign</td>
<td>Copy properties from one TField instance to another</td>
</tr>
<tr>
<td>428</td>
<td>AssignValue</td>
<td>Assign value of a variant record to the field.</td>
</tr>
<tr>
<td>429</td>
<td>Clear</td>
<td>Clear the field contents.</td>
</tr>
<tr>
<td>428</td>
<td>Create</td>
<td>Create a new TField instance</td>
</tr>
<tr>
<td>428</td>
<td>Destroy</td>
<td>Destroy the TField instance</td>
</tr>
<tr>
<td>429</td>
<td>FocusControl</td>
<td>Set focus to the first control connected to this field.</td>
</tr>
<tr>
<td>429</td>
<td>GetData</td>
<td>Get the data from this field</td>
</tr>
<tr>
<td>430</td>
<td>IsBlob</td>
<td>Is the field a BLOB field (untyped data of indeterminate size).</td>
</tr>
<tr>
<td>430</td>
<td>IsValidChar</td>
<td>Check whether a character is valid input for the field</td>
</tr>
<tr>
<td>430</td>
<td>RefreshLookupList</td>
<td>Refresh the lookup list</td>
</tr>
<tr>
<td>430</td>
<td>SetData</td>
<td>Save the field data</td>
</tr>
<tr>
<td>431</td>
<td>SetFieldType</td>
<td>Set the field data type</td>
</tr>
<tr>
<td>431</td>
<td>Validate</td>
<td>Validate the data buffer</td>
</tr>
</tbody>
</table>
## 11.30.3 Property overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Properties</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>443</td>
<td>Alignment</td>
<td>rw</td>
<td>Alignment for this field</td>
</tr>
<tr>
<td>435</td>
<td>AsAnsiString</td>
<td>rw</td>
<td>Return field contents as an ANSI string</td>
</tr>
<tr>
<td>431</td>
<td>AsBCD</td>
<td>rw</td>
<td>Access the field’s contents as a BCD (Binary coded Decimal)</td>
</tr>
<tr>
<td>432</td>
<td>AsBoolean</td>
<td>rw</td>
<td>Access the field’s contents as a Boolean value.</td>
</tr>
<tr>
<td>432</td>
<td>AsBytes</td>
<td>rw</td>
<td>Retrieve the contents of the field as an array of bytes</td>
</tr>
<tr>
<td>432</td>
<td>As Currency</td>
<td>rw</td>
<td>Access the field’s contents as a Currency value.</td>
</tr>
<tr>
<td>433</td>
<td>AsDateTime</td>
<td>rw</td>
<td>Access the field’s contents as a TDateTime value.</td>
</tr>
<tr>
<td>433</td>
<td>AsFloat</td>
<td>rw</td>
<td>Access the field’s contents as a floating-point (Double) value.</td>
</tr>
<tr>
<td>434</td>
<td>AsInteger</td>
<td>rw</td>
<td>Access the field’s contents as a 32-bit signed integer (longint) value.</td>
</tr>
<tr>
<td>434</td>
<td>AsLargeInt</td>
<td>rw</td>
<td>Access the field’s contents as a 64-bit signed integer (longint) value.</td>
</tr>
<tr>
<td>433</td>
<td>AsLongint</td>
<td>rw</td>
<td>Access the field’s contents as a 32-bit signed integer (longint) value.</td>
</tr>
<tr>
<td>434</td>
<td>AsString</td>
<td>rw</td>
<td>Access the field’s contents as an AnsiString value.</td>
</tr>
<tr>
<td>435</td>
<td>AsUnicodeString</td>
<td>rw</td>
<td>Field contents as a UnicodeString</td>
</tr>
<tr>
<td>435</td>
<td>AsUTF8String</td>
<td>rw</td>
<td>Field contents as a UTF8 String</td>
</tr>
<tr>
<td>436</td>
<td>AsVariant</td>
<td>rw</td>
<td>Access the field’s contents as a Variant value.</td>
</tr>
<tr>
<td>436</td>
<td>AsWideString</td>
<td>rw</td>
<td>Access the field’s contents as a WideString value.</td>
</tr>
<tr>
<td>437</td>
<td>AttributeSet</td>
<td>rw</td>
<td>Not used: dictionary information</td>
</tr>
<tr>
<td>437</td>
<td>Calculated</td>
<td>r</td>
<td>Is the field a calculated field ?</td>
</tr>
<tr>
<td>437</td>
<td>CanModify</td>
<td>r</td>
<td>Can the field’s contents be modified.</td>
</tr>
<tr>
<td>433</td>
<td>ConstraintErrorMessage</td>
<td>rw</td>
<td>Message to display if the <strong>CustomConstraint</strong> constraint is violated.</td>
</tr>
<tr>
<td>437</td>
<td>CurValue</td>
<td>r</td>
<td>Current value of the field</td>
</tr>
<tr>
<td>433</td>
<td>CustomConstraint</td>
<td>rw</td>
<td>Custom constraint for the field’s value</td>
</tr>
<tr>
<td>437</td>
<td>DataSet</td>
<td>rw</td>
<td>Dataset this field belongs to</td>
</tr>
<tr>
<td>438</td>
<td>DataSize</td>
<td>r</td>
<td>Size of the field’s data</td>
</tr>
<tr>
<td>438</td>
<td>DataType</td>
<td>r</td>
<td>The data type of the field</td>
</tr>
<tr>
<td>439</td>
<td>DefaultExpression</td>
<td>rw</td>
<td>Default value for the field</td>
</tr>
<tr>
<td>438</td>
<td>DisplayName</td>
<td>r</td>
<td>User-readable fieldname</td>
</tr>
<tr>
<td>438</td>
<td>DisplayText</td>
<td>r</td>
<td>Formatted field value</td>
</tr>
<tr>
<td>439</td>
<td>DisplayWidth</td>
<td>rws</td>
<td>Width of the field in characters</td>
</tr>
<tr>
<td>439</td>
<td>EditMask</td>
<td>rw</td>
<td>Specify an edit mask for an edit control</td>
</tr>
<tr>
<td>439</td>
<td>EditMaskPtr</td>
<td>r</td>
<td>Alias for EditMask</td>
</tr>
<tr>
<td>442</td>
<td>FieldDef</td>
<td>r</td>
<td>Fielddef associated with this field</td>
</tr>
<tr>
<td>445</td>
<td>FieldKind</td>
<td>rw</td>
<td>The kind of field.</td>
</tr>
<tr>
<td>445</td>
<td>FieldName</td>
<td>rw</td>
<td>Name of the field</td>
</tr>
<tr>
<td>439</td>
<td>FieldNo</td>
<td>r</td>
<td>Number of the field in the record</td>
</tr>
<tr>
<td>445</td>
<td>HasConstraints</td>
<td>r</td>
<td>Does the field have any constraints defined</td>
</tr>
<tr>
<td>446</td>
<td>ImportedConstraints</td>
<td>rw</td>
<td>Constraint for the field value on the level of the underlying database</td>
</tr>
<tr>
<td>445</td>
<td>Index</td>
<td>rw</td>
<td>Index of the field in the list of fields</td>
</tr>
<tr>
<td>440</td>
<td>IsIndexField</td>
<td>r</td>
<td>Is the field an indexed field ?</td>
</tr>
<tr>
<td>440</td>
<td>IsNull</td>
<td>r</td>
<td>Is the field empty</td>
</tr>
<tr>
<td>446</td>
<td>KeyFields</td>
<td>rw</td>
<td>Key fields to use when looking up a field value.</td>
</tr>
<tr>
<td>447</td>
<td>Lookup</td>
<td>rws</td>
<td>Is the field a lookup field</td>
</tr>
<tr>
<td>446</td>
<td>LookupCache</td>
<td>rw</td>
<td>Should lookup values be cached</td>
</tr>
<tr>
<td>446</td>
<td>LookupDataSet</td>
<td>rw</td>
<td>Dataset with lookup values</td>
</tr>
<tr>
<td>447</td>
<td>LookupKeyFields</td>
<td>rw</td>
<td>Names of fields on which to perform a locate</td>
</tr>
<tr>
<td>442</td>
<td>LookupList</td>
<td>r</td>
<td>List of lookup values</td>
</tr>
<tr>
<td>447</td>
<td>LookupResultField</td>
<td>rw</td>
<td>Name of field to use as lookup value</td>
</tr>
<tr>
<td>440</td>
<td>NewValue</td>
<td>rw</td>
<td>The new value of the field</td>
</tr>
<tr>
<td>440</td>
<td>Offset</td>
<td>r</td>
<td>Offset of the field’s value in the dataset buffer</td>
</tr>
<tr>
<td>442</td>
<td>OldValue</td>
<td>r</td>
<td>Old value of the field</td>
</tr>
<tr>
<td>449</td>
<td>OnChange</td>
<td>rw</td>
<td>Event triggered when the field’s value has changed</td>
</tr>
</tbody>
</table>
11.30.4 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 method will create the necessary TField instances, if none had been created in the designer.

See also: TDataset.Open

11.30.5 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

11.30.6 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

11.30.7 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 (338) exception.

See also: TField.Assign (428), TField.Value (442)

11.30.8 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 (440) 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 (338) exception.

See also: TField.IsNull (440), TField.Value (442)

11.30.9 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 (385), TDataset.DisableControls (384)

11.30.10 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 (440), TField.SetData (430), TField.Value (442)
### 11.30.11 TField.IsBlob

**Synopsis:** Is the field a BLOB field (untyped data of indeterminate size).

**Declaration:**
```plaintext
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` (352)

### 11.30.12 TField.IsValidChar

**Synopsis:** Check whether a character is valid input for the field

**Declaration:**
```plaintext
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` (441) property. The `ValidChars` property will be initialized to a correct set of characters by descendant 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` (441)

### 11.30.13 TField.RefreshLookupList

**Synopsis:** Refresh the lookup list

**Declaration:**
```plaintext
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` (446) 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` (446), `LookupKeyFields` (447), `LookupResultField` (447)

### 11.30.14 TField.SetData

**Synopsis:** Save the field data

**Declaration:**
```plaintext
procedure SetData(Buffer: Pointer); Overload
procedure SetData(Buffer: Pointer; NativeFormat: Boolean); Overload
```

**Visibility:** public
CHAPTER 11. REFERENCE FOR UNIT 'DB'

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 (429)**, **TField.Value (442)**

11.30.15 **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 (438)**

11.30.16 **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 (450)** event handler, if one is set, to allow the application programmer to program additional checks.

See also: **TField.SetData (430)**, **TField.OnValidate (450)**

11.30.17 **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). Therefore, 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 (432)**, **TField.Value (442)**
**11.30.18 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). Therefore, 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 (442), TField.AsInteger (434)

**11.30.19 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 (351)

**11.30.20 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). Therefore, 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 (442), TField.AsFloat (433)
11.30.21 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). Therefore, 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 (442), TField.AsString (434)

11.30.22 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). Therefore, 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 (442), TField.AsString (434), TField.AsCurrency (432)

11.30.23 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). Therefore, 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 (434).

See also: TField.Value (442), TField.AsString (434), TField.AsInteger (434)
11.30.24 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). Therefore, 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 (442), TField.AsString (434), TField.AsInteger (434)

11.30.25 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 an 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). Therefore, 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 (442), TField.AsString (434), TField.AsLongint (433), TField.AsInt64 (424)

11.30.26 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 an 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). Therefore, 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 (442), TField.AsWideString (436)
11.30.27  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 (434), TField.AsUnicodeString (435), TField.AsUTF8String (435), TField.CodePage (424)

11.30.28  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 (424) 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 (434), TField.AsAnsiString (435), TField.AsUTF8String (435), TField.CodePage (424)

11.30.29  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 (424). 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 (434), TField.AsUnicodeString (435), TField.AsAnsi8String (424), TField.CodePage (424)
11.30.30 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). Therefore, 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 (442), TField.Astring (424)

11.30.31 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). Therefore, 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 (442), TField.Astring (424)

11.30.32 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.
11.30.33 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 (445) 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 (445)

11.30.34 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 (448), TDataSet.CanModify (396)

11.30.35 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 (442)

11.30.36 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 (375), TDataSet.Fields (401)
11.30.37 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 property which declares a logical size for datatypes that have a variable size (such as string fields). For BLOB fields, use the TBlobField.BlobSize property to get the size of the field’s contents for the current record.

See also: TField.Size, TBlobField.BlobSize

11.30.38 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

11.30.39 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 property, unless assigned another value. The use of this property is deprecated. Use DisplayLabel instead.

See also: TField.FieldName

11.30.40 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 (434) property, which does not take into account any formatting.

See also: TField.AsString (434)

11.30.41 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 (418), TStringField/EditMask (510)

11.30.42 TField.EditMaskPtr
Synopsis: Alias for EditMask
Declaration: Property EditMaskPtr : TEditMask
  Visibility: public
  Access: Read
Description: EditMaskPtr is a read-only alias for the EditMask (439) property. It is not used.
  See also: TField/EditMask (439)

11.30.43 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 (445)
**11.30.44 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, descendants of TDataset (375) can change this to True.

See also: TField.Calculated (437)

**11.30.45 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 (389) is called or Clear (429) is called then IsNull will return True.

See also: TField.Clear (429), TDataset.Insert (389)

**11.30.46 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 (375) does not yet support this.

See also: TField.Value (442), TField.CurValue (437)

**11.30.47 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 (439), TField.Index (445), TField.Datasize (438)
11.30.48  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 (438) should be used. For blob fields, the current size of the data is not guaranteed to be present.

See also: DataSize (438)

11.30.49  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 (449) event handler if a handler was specified. For display purposes, the TField.DisplayText (438) 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 (434) property, this will invoke the TField.OnSetText (449) event handler, if one is set.

See also: TField.AsString (434), TField.DisplayText (438), TField.Value (442)

11.30.50  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 (430) function.

See also: TField.IsValidChar (430)
11.30.51 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 (440) 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 (440), TField.Text (441), TField.DisplayText (438)

11.30.52 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 (442), TField.CurValue (437), TField.NewValue (440)

11.30.53 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 (446) property is set to True. It can be refreshed using the RefreshLookupList (430) method.

See also: TField.RefreshLookupList (430), TField.LookupCache (446)

11.30.54 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` (398) 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` (398)

### 11.30.55 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` (438)

### 11.30.56 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` (443) as a message.

This feature is not yet implemented in FPC.

See also: `TField.ConstraintErrorMessage` (443), `TField.ImportedConstraint` (446)

### 11.30.57 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 (443) is violated. This feature is not yet implemented in FPC.

See also: TField.CustomConstraint (443)

11.30.58 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 (380) or TDataset.Insert (389) 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 (389), TDataset.Append (380), TDataset.CustomConstraint (375)

11.30.59 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 (445) property, unless assigned another value.

See also: TField.FieldName (445)

11.30.60 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 (441) for string fields) but can be modified to obtain a more appropriate value for the field’s expected content.

See also: TField.Alignment (443), TField.DisplayText (438)
11.30.61 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.

11.30.62 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 (450), TField.Size (441), TField.DataType (438)

11.30.63 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 (443) or ImportedConstraint (446) properties is set to a non-empty value.

See also: CustomConstraint (443), ImportedConstraint (446)

11.30.64 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 (439) property should be used for that.

See also: TField.FieldNo (439)
11.30.65  **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 (375) descendent.

See also: TField.CustomConstraint (443), TDataSet (375), TField.ConstraintErrorMessage (443)

11.30.66  **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 (447) in the dataset pointed to by the LookupDataset (446) property.

See also: LookupKeyFields (447), LookupDataset (446)

11.30.67  **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 (447) in the dataset pointed to by the LookupDataset (446) property. The list of key, value pairs is available through the TField.LookupList (442) property.

See also: LookupKeyFields (447), LookupDataset (446), TField.LookupList (442)

11.30.68  **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 (447) property is used as a list of fields to locate a record in this dataset, and the value of the LookupResultField (447) field is then used as the value of the lookup field.

See also: KeyFields (446), LookupKeyFields (447), LookupResultField (447), LookupCache (446)
11.30.69 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 (446) property. These fields will be used when locating a record corresponding to the values in the TField.KeyFields (446) property.
See also: KeyFields (446), LookupDataset (446), LookupResultField (447), LookupCache (446)

11.30.70 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 (446) 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 (446), LookupDataset (446), LookupKeyFields (447), LookupCache (446)

11.30.71 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 (445) equals fkLookup, False otherwise. Setting the Lookup property will switch the FieldKind between the fkLookup and fkData.
See also: TField.FieldKind (445)

11.30.72 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 (375) descendent or the database engine support retrieval of this property. It can be used to automatically create update statements, together with the TField.ProviderFlags (448) property.
See also: TDataset (375), TField.ProviderFlags (448)
**11.30.73 TField.ProviderFlags**

**Synopsis:** Flags for provider or update support

**Declaration:**

```pascal
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 (447)

**11.30.74 TField.ReadOnly**

**Synopsis:** Is the field read-only

**Declaration:**

```pascal
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 (454)

**11.30.75 TField.Required**

**Synopsis:** Does the field require a value

**Declaration:**

```pascal
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 (375) 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 (392), ReadOnly (448), Visible (449)
11.30.76 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 (448), TField.Required (448)

11.30.77 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 (450) 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 (449)

11.30.78 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 (441) or TField.DisplayText (438) 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 (441), TField.DisplayText (438), TField.OnSetText (449), TFieldGetTextEvent (326)

11.30.79 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` (441) 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` (449) handler, if it exists.

See also: `TField.Text` (441), `TField.OnGetText` (449), `TFieldGetTextEvent` (326)

### 11.30.80 TField.OnValidate

**Synopsis:** Event to validate the value of a field before it is written to the data buffer

**Declaration:**

```pascal
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` (441), `TField.OnGetText` (449), `TField.OnSetText` (449), `TField.OnChange` (449)

### 11.31 TFieldDef

#### 11.31.1 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` (398), `TFieldDefs` (455)

#### 11.31.2 Method overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>451</td>
<td>Assign</td>
<td>Assign the contents of one <code>TFieldDef</code> instance to another.</td>
</tr>
<tr>
<td>451</td>
<td>Create</td>
<td>Constructor for <code>TFieldDef</code>.</td>
</tr>
<tr>
<td>452</td>
<td>CreateField</td>
<td>Create <code>TField</code> instance based on definitions in current <code>TFieldDef</code> instance.</td>
</tr>
<tr>
<td>451</td>
<td>Destroy</td>
<td>Free the <code>TFieldDef</code> instance</td>
</tr>
</tbody>
</table>
### 11.31.3 Property overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Properties</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>454</td>
<td>Attributes</td>
<td>rw</td>
<td>Additional attributes of the field.</td>
</tr>
<tr>
<td>453</td>
<td>CharSize</td>
<td>r</td>
<td>Character size</td>
</tr>
<tr>
<td>453</td>
<td>Codepage</td>
<td>r</td>
<td>System code page for the values in string and wide string field types</td>
</tr>
<tr>
<td>454</td>
<td>DataType</td>
<td>rw</td>
<td>Data type for the field</td>
</tr>
<tr>
<td>452</td>
<td>FieldClass</td>
<td>r</td>
<td>TField class used for this fielddef</td>
</tr>
<tr>
<td>452</td>
<td>FieldNo</td>
<td>r</td>
<td>Field number</td>
</tr>
<tr>
<td>453</td>
<td>InternalCalcField</td>
<td>rw</td>
<td>Is this a definition of an internally calculated field?</td>
</tr>
<tr>
<td>454</td>
<td>Precision</td>
<td>rw</td>
<td>Precision used in BCD (Binary Coded Decimal) fields</td>
</tr>
<tr>
<td>453</td>
<td>Required</td>
<td>rw</td>
<td>Is the field required?</td>
</tr>
<tr>
<td>455</td>
<td>Size</td>
<td>rw</td>
<td>Size of the buffer needed to store the data of the field</td>
</tr>
</tbody>
</table>

### 11.31.4 TFieldDef.Create

**Synopsis:** Constructor for TFieldDef.

**Declaration:**

```pascal
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 (450), datatype (454), size (455), precision (454), FieldNo (452), Required (453) and CodePage (453) property are all set according to the passed arguments.

**Errors:** If a duplicate name is passed, then an exception will occur.

**See also:** Name (450), datatype (454), size (455), precision (454), FieldNo (452), Required (453), CodePage (453)

### 11.31.5 TFieldDef.Destroy

**Synopsis:** Free the TFieldDef instance

**Declaration:**

```pascal
destructor Destroy; Override
```

**Visibility:** public

**Description:** Destroy destroys the TFieldDef instance. It simply calls the inherited destructor.

**See also:** TFieldDef.Create (451)

### 11.31.6 TFieldDef.Assign

**Synopsis:** Assign the contents of one TFieldDef instance to another.

**Declaration:**

```pascal
procedure Assign(APersistent: TPersistent); Override
```

451
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 (450), then an exception will be raised.

11.31.7 TFieldDef.CreateField

Synopsis: Create TField instance based on definitions in current TFieldDef instance.

Declaration: function CreateField(AOwner: TComponent) : TField

Visibility: public

Description: CreateField determines, based on the DataType (454) what TField (424) descendent it should create, and then returns a newly created instance of this class. It sets the appropriate defaults for the Size (441), FieldName (445), FieldNo (439), Precision (424), ReadOnly (448) and Required (448) 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 (320) 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 (320), TField (424)

11.31.8 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 (452) 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 (375), CreateField (452), TField (424)

11.31.9 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.
11.31.10 TFieldDef.CharSize

Synopsis: Character size

Declaration: Property CharSize : Word

Visibility: public
Access: Read

Description: CharSize is only relevant for string fields: it indicates the number of bytes used to represent a single character. It is calculated from the TFieldDef.CodePage (453) 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 (453), TFieldDef.Size (455)

11.31.11 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.

11.31.12 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.

11.31.13 TFieldDef.Codepage

Synopsis: System code page for the values in string and wide string field types

Declaration: Property Codepage : TSystemCodePage

Visibility: public
Access: Read
CHAPTER 11. REFERENCE FOR UNIT 'DB'

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 (451), TFieldDef.DataType (454)

### 11.31.14 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 (453)

### 11.31.15 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 (452), TFieldDef.CreateField (452)

### 11.31.16 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 (455)
11.31.17 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 (454), TFieldDef.DataType (454)

### 11.32 TFieldDefs

#### 11.32.1 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 descendant 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 (450), TDataset (375)

#### 11.32.2 Method overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>456</td>
<td>Add</td>
<td>Add a new field definition to the collection.</td>
</tr>
<tr>
<td>456</td>
<td>AddFieldDef</td>
<td>Add new TFieldDef</td>
</tr>
<tr>
<td>456</td>
<td>Assign</td>
<td>Copy all items from one dataset to another</td>
</tr>
<tr>
<td>455</td>
<td>Create</td>
<td>Create a new instance of TFieldDefs</td>
</tr>
<tr>
<td>457</td>
<td>Find</td>
<td>Find item by name</td>
</tr>
<tr>
<td>457</td>
<td>MakeNameUnique</td>
<td>Create a unique field name starting from a base name</td>
</tr>
<tr>
<td>457</td>
<td>Update</td>
<td>Force update of definitions</td>
</tr>
</tbody>
</table>

#### 11.32.3 Property overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Properties</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>457</td>
<td>HiddenFields</td>
<td>rw</td>
<td>Should field instances be created for hidden fields</td>
</tr>
<tr>
<td>458</td>
<td>Items</td>
<td>rw</td>
<td>Indexed access to the fielddef instances</td>
</tr>
</tbody>
</table>

#### 11.32.4 TFieldDefs.Create

**Synopsis:** Create a new instance of TFieldDefs
CHAPTER 11. REFERENCE FOR UNIT 'DB'

Declaration: constructor Create(ADataset: TDataSet)

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 (450), TDataset (375)

11.32.5 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

procedure Add(const AName: string; ADataType: TFieldType; ASize: Word; 
ARequired: Boolean); 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 (456)

11.32.6 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 (456)

11.32.7 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 (451)
11.32.8  TFieldDefs.Find

Synopsis:  Find item by name

Declaration:  function Find(const AName: string) : TFieldDef

Visibility:  public

Description:  Find simply calls the inherited TDefCollection.Find (422) to find an item with name AName and typecasts the result to TFieldDef.

See also:  TDefCollection.Find (422), TNamedItem.Name (487)

11.32.9  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 (423)

11.32.10  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 (457), TFieldDef.Name (450)

11.32.11  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 (454)
11.32.12  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 (450)

11.33  TFields

11.33.1  Description

TFields mimics a TCollection class for the Fields (401) property of TDataSet (375) instance. Since TField (424) is a descendent of TComponent, it cannot be an item of a collection, and must be managed by another class.

See also: TField (424), TDataSet (375), TDataSet.Fields (401)

11.33.2  Method overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>459</td>
<td>Add</td>
<td>Add a new field to the list</td>
</tr>
<tr>
<td>459</td>
<td>CheckFieldName</td>
<td>Check field name for duplicate entries</td>
</tr>
<tr>
<td>459</td>
<td>CheckFieldNames</td>
<td>Check a list of field names for duplicate entries</td>
</tr>
<tr>
<td>460</td>
<td>Clear</td>
<td>Clear the list of fields</td>
</tr>
<tr>
<td>458</td>
<td>Create</td>
<td>Create a new instance of TFields</td>
</tr>
<tr>
<td>459</td>
<td>Destroy</td>
<td>Free the TFields instance</td>
</tr>
<tr>
<td>460</td>
<td>FieldByName</td>
<td>Find a field based on its name</td>
</tr>
<tr>
<td>460</td>
<td>FieldByNumber</td>
<td>Search field based on its fieldnumber</td>
</tr>
<tr>
<td>460</td>
<td>FindField</td>
<td>Find a field based on its name</td>
</tr>
<tr>
<td>461</td>
<td>GetEnumerator</td>
<td>Return an enumerator for the for..in construct</td>
</tr>
<tr>
<td>461</td>
<td>GetFieldNames</td>
<td>Get the list of fieldnames</td>
</tr>
<tr>
<td>461</td>
<td>IndexOf</td>
<td>Return the index of a field instance</td>
</tr>
<tr>
<td>461</td>
<td>Remove</td>
<td>Remove an instance from the list</td>
</tr>
</tbody>
</table>

11.33.3  Property overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Properties</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>462</td>
<td>Count</td>
<td>r</td>
<td>Number of fields in the list</td>
</tr>
<tr>
<td>462</td>
<td>Dataset</td>
<td>r</td>
<td>Dataset the fields belong to</td>
</tr>
<tr>
<td>462</td>
<td>Fields</td>
<td>rw</td>
<td>Indexed access to the fields in the list</td>
</tr>
</tbody>
</table>

11.33.4  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 (462) property, and initializes an internal list object to store the list of fields.

See also: TDataset (375), TFields.Dataset (462), TField (424)

### 11.33.5 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 (462). After that it cleans up the internal structures and then calls the inherited destructor.

See also: TDataset (375), TField (424), TFields.Clear (460)

### 11.33.6 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 (424) 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 (424), TFields.Clear (460)

### 11.33.7 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 (458)). If it does, then an EDatabaseError (338) exception is raised.

See also: TField.FieldName (445), TFields.Find (458)

### 11.33.8 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 (459).
CHAPTER 11. REFERENCE FOR UNIT 'DB'

Errors: Spaces are not discarded, so leaving a space after or before a fieldname will not find the fieldname, and will yield a false negative result.

See also: TField.FieldName (445), TFields.CheckFieldName (459), TFields.Find (458)

11.33.9 TFields.Clear

Synopsis: Clear the list of fields

Declaration: procedure Clear

Visibility: public

Description: Clear removes all TField (424) var instances from the list. All field instances are freed after they have been removed from the list.

See also: TField (424)

11.33.10 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 (445) property matches Value. The search is performed case-insensitively. If no field instance is found, then Nil is returned.

See also: TFields.FieldByName (460)

11.33.11 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 (445) property matches Value. The search is performed case-insensitively. If no field instance is found, then Nil is returned.

Errors: If no field instance is found, then an exception is raised. If this behaviour is undesired, use TField.FindField (424), where Nil is returned if no match is found.

See also: TFields.FindField (460), TFields.FieldName (458), Tfields.FieldByNumber (460), TFields.IndexOf (461)

11.33.12 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 (439) property matches the FieldNo parameter. If no such field is found, Nil is returned.

See also: TFields.FieldByName (460), TFields.FindField (460), TFields.IndexOf (461)
11.33.13 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 (462)

See also: TFieldsEnumerator (462), #rtl.system.IEnumerable (??)

11.33.14 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 (445)

11.33.15 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 (460), TFields.FieldByNumber (460), TFields.FindField (460)

11.33.16 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 (460)
11.33.17  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 (462) property are numbered from 0 to Count-1.

See also: TFields.fields (462)

11.33.18  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 (458) 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 (458), TField.Dataset (437), TDataset (375)

11.33.19  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 (460)

11.34  TFieldsEnumerator

11.34.1  Description

TFieldsEnumerator implements all the methods of IEnumerator so a TFields (458) 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 (424), TFields (458), #rtl.system.IEnumerator (??)

### 11.34.2 Method overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>463</td>
<td>Create</td>
<td>Create a new instance of TFieldsEnumerator.</td>
</tr>
<tr>
<td>463</td>
<td>MoveNext</td>
<td>Move the current field to the next field in the collection.</td>
</tr>
</tbody>
</table>

### 11.34.3 Property overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Properties</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>464</td>
<td>Current</td>
<td>r</td>
<td>Return the current field</td>
</tr>
</tbody>
</table>

### 11.34.4 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 (458) instance that created the enumerator. It initializes the enumerator position.

### 11.34.5 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 (464)
11.34.6 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 (463)

11.35 TFloatField

11.35.1 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 (465), MaxValue (465), MinValue (466) and overrides some TField (424) 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 (465), MaxValue (465), MinValue (466)

11.35.2 Method overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>465</td>
<td>CheckRange</td>
<td>Check whether a value is in the allowed range of values for the field</td>
</tr>
<tr>
<td>464</td>
<td>Create</td>
<td>Create a new instance of the TFloatField</td>
</tr>
</tbody>
</table>

11.35.3 Property overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Properties</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>465</td>
<td>Currency</td>
<td>rw</td>
<td>Is the field a currency field.</td>
</tr>
<tr>
<td>465</td>
<td>MaxValue</td>
<td>rw</td>
<td>Maximum value for the field</td>
</tr>
<tr>
<td>466</td>
<td>MinValue</td>
<td>rw</td>
<td>Minimum value for the field</td>
</tr>
<tr>
<td>466</td>
<td>Precision</td>
<td>rw</td>
<td>Precision (number of digits) of the field in text representations</td>
</tr>
<tr>
<td>465</td>
<td>Value</td>
<td>rw</td>
<td>Value of the field as a double type</td>
</tr>
</tbody>
</table>

11.35.4 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.
11.35.5 **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 (466) and MaxValue (465) properties. If the value lies outside of the allowed range, then False is returned.

See also: MaxValue (465), MinValue (466)

11.35.6 **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 (433)

See also: TField.AsFloat (433), TField.Value (442)

11.35.7 **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 (438) and TField.Text (441) 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 (438), TField.Text (441), TNumericField.DisplayFormat (488), TNumericField.EditFormat (488)

11.35.8 **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 (466) should also be set, because it will also be checked.

See also: TFloatField.MaxValue (466)

11.35.9 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 (465) should also be set, because it will also be checked.

See also: TFloatField.MaxValue (465), TFloatField.CheckRange (465)

11.35.10 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 (438) or TField.Text (441), it is used in the arguments to FormatFloat (??).

See also: TField.Displaytext (438), TField.Text (441), FormatFloat (??)

11.36 TFMTBCDField

11.36.1 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 (464)
11.36.2 Method overview

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CheckRange</td>
<td>Check value if it is in the range defined by MinValue and MaxValue</td>
</tr>
<tr>
<td>Create</td>
<td>Create a new instance of the TFMTBCDField class.</td>
</tr>
</tbody>
</table>

11.36.3 Property overview

<table>
<thead>
<tr>
<th>Properties</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Currency</td>
<td>rw</td>
<td>Does the field contain currency data?</td>
</tr>
<tr>
<td>MaxValue</td>
<td>rw</td>
<td>Maximum value for the field</td>
</tr>
<tr>
<td>MinValue</td>
<td>rw</td>
<td>Minimum value for the field</td>
</tr>
<tr>
<td>Precision</td>
<td>rw</td>
<td>Total number of digits in the BCD data</td>
</tr>
<tr>
<td>Size</td>
<td></td>
<td>Number of digits after the decimal point</td>
</tr>
<tr>
<td>Value</td>
<td>rw</td>
<td>The value of the field as a BCD value</td>
</tr>
</tbody>
</table>

11.36.4 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 (319), MaxValue (319), Size (441) (15) and Precision (319) (2) fields to their default values.

**See also:** MinValue (319), MaxValue (319), Size (441), Precision (319)

11.36.5 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 (319) and MaxValue (319) 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 (319), MaxValue (319)

11.36.6 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 (433), TField.AsCurrency (432)
11.36.7  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 (433), TField.AsCurrency (432), Size (319)

11.36.8  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 (424) or EditFormat (424) are set, these values are used instead to format the value.

See also: TField.DisplayFormat (424), TField.EditFormat (424)

11.36.9  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 (319) or it will not have any effect.

See also: TFMTBCDField.CheckRange (467), MinValue (319)

11.36.10  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 (319) or it will not have any effect.

See also: TFMTBCDField.CheckRange (467), MaxValue (319)
11.36.11  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 property it determines the maximum allowed range of values for the field. This range can be restricted using the MinValue and MaxValue properties.
See also: MinValue, MaxValue, Precision

11.37  TGraphicField

11.37.1  Description
TGraphicField is the class used when a dataset must manage graphical BLOB data. (TField.DataType equals ftGraphic). It initializes some of the properties of the TField class. All methods to be able to work with graphical BLOB data have been implemented in the TBlobField 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, TField, TBLOBField, TMemoField, TWideMemoField

11.37.2  Method overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>469</td>
<td>Create</td>
<td>Create a new instance of the TGraphicField class</td>
</tr>
</tbody>
</table>

11.37.3  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 properties to configure the instance for working with graphical BLOB values.
See also: TField

11.38  TGuidField

11.38.1  Description
TGUIDField is the class used when a dataset must manage native variant-typed data. (TField.DataType equals ftGUID). It initializes some of the properties of the TField 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 (375), TField (424), TGuidField.AsGuid (470)

### 11.38.2 Method overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>470</td>
<td>Create</td>
<td>Create a new instance of the TGUIDField class</td>
</tr>
</tbody>
</table>

### 11.38.3 Property overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Properties</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>470</td>
<td>AsGuid</td>
<td>rw</td>
<td>Field content as a GUID value</td>
</tr>
</tbody>
</table>

### 11.38.4 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 (424) properties to configure the instance for working with GUID values.

See also: TField (424)

### 11.38.5 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: TFieldAsString (434)

### 11.39 TIndexDef

### 11.39.1 Description

TIndexDef describes one index in a set of indexes of a TDataset (375) instance. The collection of indexes is described by the TIndexDefs (473) class. It just has the necessary properties to describe an index, but does not implement any functionality to maintain an index.

See also: TIndexDefs (473)
11.39.2 Method overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>471</td>
<td>Create</td>
<td>Create a new index definition</td>
</tr>
</tbody>
</table>

11.39.3 Property overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Properties</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>472</td>
<td>CaseInsFields</td>
<td>rw</td>
<td>Fields in field list that are ordered case-insensitively</td>
</tr>
<tr>
<td>472</td>
<td>DescFields</td>
<td>rw</td>
<td>Fields in field list that are ordered descending</td>
</tr>
<tr>
<td>471</td>
<td>Expression</td>
<td>rw</td>
<td>Expression that makes up the index values</td>
</tr>
<tr>
<td>471</td>
<td>Fields</td>
<td>rw</td>
<td>Fields making up the index</td>
</tr>
<tr>
<td>472</td>
<td>Options</td>
<td>rw</td>
<td>Index options</td>
</tr>
<tr>
<td>473</td>
<td>Source</td>
<td>rw</td>
<td>Source of the index</td>
</tr>
</tbody>
</table>

11.39.4 TIndexDef.Create

**Synopsis:** Create a new index definition

**Declaration:**

```pascal
constructor Create(Owner: TIndexDefs; const AName: string;
const TheFields: string; TheOptions: TIndexOptions)
; Overload
```

**Visibility:** public

**Description:** Create initializes a new TIndexDef (470) 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 (473) 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 (473), TIndexDef.Options (472), TIndexDef.Fields (471)

11.39.5 TIndexDef.Expression

**Synopsis:** Expression that makes up the index values

**Declaration:**

```pascal
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 (472)

**See also:** TIndexDef.Options (472), TIndexDef.Fields (471)

11.39.6 TIndexDef.Fields

**Synopsis:** Fields making up the index

**Declaration:**

```pascal
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 (472) property can be used to determine the fields in the list that have a descending sort order. The TIndexDef.CaseInsFields (472) property determines which fields are sorted in a case-insensitive manner.

See also: TIndexDef.DescFields (472), TIndexDef.CaseInsFields (472), TIndexDef.Expression (471)

11.39.7 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 (471) property which are ordered in a case-insensitive manner. CaseInsFields may not contain fieldnames that do not appear in Fields.

See also: TIndexDef.Fields (471), TIndexDef.Expression (471), TIndexDef.DescFields (472)

11.39.8 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 (471) property which are ordered in a descending manner. DescFields may not contain fieldnames that do not appear in Fields.

See also: TIndexDef.Fields (471), TIndexDef.Expression (471), TIndexDef.DescFields (472)

11.39.9 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 (330) for more information on the various available options.

See also: TIndexOptions (330)
11.39.10  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.

11.40  TIndexDefs

11.40.1  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 (470), TIndexDefs.Items (475)

11.40.2  Method overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>474</td>
<td>Add</td>
<td>Add a new index definition with given name and options</td>
</tr>
<tr>
<td>474</td>
<td>AddIndexDef</td>
<td>Add a new, empty, index definition</td>
</tr>
<tr>
<td>473</td>
<td>Create</td>
<td>Create a new TIndexDefs instance</td>
</tr>
<tr>
<td>474</td>
<td>Find</td>
<td>Find an index by name</td>
</tr>
<tr>
<td>474</td>
<td>FindIndexForFields</td>
<td>Find index definition based on field names</td>
</tr>
<tr>
<td>475</td>
<td>GetIndexForFields</td>
<td>Get index definition based on field names</td>
</tr>
<tr>
<td>475</td>
<td>Update</td>
<td>Called whenever one of the items changes</td>
</tr>
</tbody>
</table>

11.40.3  Property overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Properties</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>475</td>
<td>Items</td>
<td>rw</td>
<td>Indexed access to the index definitions</td>
</tr>
</tbody>
</table>

11.40.4  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 (470).

See also: TIndexDef (470), TIndexDefs.Destroy (473)
11.40.5  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 (470) 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 (470), TNamedItem.Name (487), TIndexDef.Fields (471), TIndexDef.Options (472),
TIndexDefs.AddIndexDef (474)

11.40.6  TIndexDefs.AddIndexDef
Synopsis: Add a new, empty, index definition
Declaration: function AddIndexDef : TIndexDef
Visibility: public
Description: AddIndexDef adds a new TIndexDef (470) 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 (474)

11.40.7  TIndexDefs.Find
Synopsis: Find an index by name
Declaration: function Find(const IndexName: string) : TIndexDef
Visibility: public
Description: Find overloads the TDefCollection.Find (422) method to search and return a TIndexDef (470)
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 (470), TDefCollection.Find (422), TIndexDefs.FindIndexForFields (474)

11.40.8  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 (471)
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 function-
ality, where Find usually returns Nil if nothing is found.
See also: TIndexDef (470), TIndexDefs.Find (474), TIndexDefs.GetIndexForFields (475)

474
11.40.9  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 property matches the list of fields in Fields. If CaseInsensitive 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 (470), TIndexDefs.Find (474), TIndexDefs.FindIndexForFields (474)

11.40.10  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.

11.40.11  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 (470)

11.41  TIntegerField

11.41.1  Description

TIntegerField is an alias for TLongIntField (477).

See also: TLongIntField (477), TField (424)

11.42  TLargeIntField

11.42.1  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 (424) 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 (477) and MaxValue (477).

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 (424), MinValue (477), MaxValue (477)

### 11.42.2 Method overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>476</td>
<td>CheckRange</td>
<td>Check whether a values falls within the allowed range</td>
</tr>
<tr>
<td>476</td>
<td>Create</td>
<td>Create a new instance of the TLargeIntField class</td>
</tr>
</tbody>
</table>

### 11.42.3 Property overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Properties</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>477</td>
<td>MaxValue</td>
<td>rw</td>
<td>Maximum value for the field</td>
</tr>
<tr>
<td>477</td>
<td>MinValue</td>
<td>rw</td>
<td>Minimum value for the field</td>
</tr>
<tr>
<td>476</td>
<td>Value</td>
<td>rw</td>
<td>Field contents as a 64-bit integer value</td>
</tr>
</tbody>
</table>

### 11.42.4 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 (424) and MinValue (477) and MaxValue (477).

See also: TField (424), MinValue (477), MaxValue (477)

### 11.42.5 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 (477) and MaxValue (477) properties. If the value lies outside of the allowed range, then False is returned.

See also: MaxValue (477), MinValue (477)

### 11.42.6 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 (434).

See also: TField.Value (442), TField.AsLargeInt (434)

11.42.7 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 (477) should also be set, because it will also be checked.

See also: TLargeIntField.MinValue (477)

11.42.8 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 (477) should also be set, because it will also be checked.

See also: TLargeIntField.MaxValue (477)

11.43 TLongIntField

11.43.1 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 (424) 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 (479) and MaxValue (479).

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 (424), MaxValue (479), MinValue (479)
CHAPTER 11. REFERENCE FOR UNIT 'DB'

11.43.2 Method overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>478</td>
<td>CheckRange</td>
<td>Check whether a valid is in the allowed range of values for the field</td>
</tr>
<tr>
<td>478</td>
<td>Create</td>
<td>Create a new instance of TLongintField</td>
</tr>
</tbody>
</table>

11.43.3 Property overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Properties</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>479</td>
<td>MaxValue</td>
<td>rw</td>
<td>Maximum value for the field</td>
</tr>
<tr>
<td>479</td>
<td>MinValue</td>
<td>rw</td>
<td>Minimum value for the field</td>
</tr>
<tr>
<td>478</td>
<td>Value</td>
<td>rw</td>
<td>Value of the field as longint</td>
</tr>
</tbody>
</table>

11.43.4 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 (479) and MaxValue (479) properties.

See also: TField (424), MaxValue (479), MinValue (479)

11.43.5 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 (479) and MaxValue (479) properties. If the value lies outside of the allowed range, then False is returned.

See also: MaxValue (479), MinValue (479)

11.43.6 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 (434) property.

See also: TField.Value (442)

478
11.43.7 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 (479)

11.43.8 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 (479)

11.44 TLookupList

11.44.1 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 (446)

11.44.2 Method overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>480</td>
<td>Add</td>
<td>Add a key, value pair to the list</td>
</tr>
<tr>
<td>480</td>
<td>Clear</td>
<td>Remove all key, value pairs from the list</td>
</tr>
<tr>
<td>480</td>
<td>Create</td>
<td>Create a new instance of TLookupList.</td>
</tr>
<tr>
<td>480</td>
<td>Destroy</td>
<td>Free a TLookupList instance from memory</td>
</tr>
<tr>
<td>480</td>
<td>FirstKeyByValue</td>
<td>Find the first key that matches a value</td>
</tr>
<tr>
<td>481</td>
<td>ValueByKey</td>
<td>Look up value based on a key</td>
</tr>
<tr>
<td>481</td>
<td>ValuesToStrings</td>
<td>Convert values to stringlist</td>
</tr>
</tbody>
</table>
11.44.3 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 (480)

11.44.4 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 (480)

11.44.5 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 (480)

11.44.6 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 (480)

11.44.7 TLookupList.FirstKeyByValue
Synopsis: Find the first key that matches a value

Declaration: function FirstKeyByValue(const AValue: Variant) : Variant

Visibility: public
**CHAPTER 11. REFERENCE FOR UNIT 'DB'**

**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 (481)

### 11.44.8 TLookupList.ValueOfKey

**Synopsis:** Look up value based on a key

**Declaration:**
```pascal
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 (480), TLookupList.Add (480)

### 11.44.9 TLookupList.ValuesToStrings

**Synopsis:** Convert values to stringlist

**Declaration:**
```pascal
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 dropdown list.

**See also:** TLookupList.ValueOfKey (481)

### 11.45 TMasterDataLink

#### 11.45.1 Description

TMasterDataLink is a TDatalink descendent which handles master-detail relations. It can be used in TDataset (375) 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 (373) property.

The class also provides functionality for keeping a list of fields that make up the master-detail functionality, in the TMasterDatalink.FieldNames (482) and TMasterDataLink.Fields (483) properties.

This class should never be used in application code.

**See also:** TDataset (375), TDatalink.DataSource (373), TDatalink.DataSet (373), TMasterDatalink.FieldNames (482), TMasterDataLink.Fields (483)

#### 11.45.2 Method overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>482</td>
<td>Create</td>
<td>Create a new instance of TMasterDataLink</td>
</tr>
<tr>
<td>482</td>
<td>Destroy</td>
<td>Free the datalink instance from memory</td>
</tr>
</tbody>
</table>

481
11.45.3 Property overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Properties</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>482</td>
<td>FieldNames</td>
<td>rw</td>
<td>List of fieldnames that make up the master-detail relationship</td>
</tr>
<tr>
<td>483</td>
<td>Fields</td>
<td>r</td>
<td>List of fields as specified in FieldNames</td>
</tr>
<tr>
<td>483</td>
<td>OnMasterChange</td>
<td>rw</td>
<td>Called whenever the master dataset data changes</td>
</tr>
<tr>
<td>483</td>
<td>OnMasterDisable</td>
<td>rw</td>
<td>Called whenever the master dataset is disabled</td>
</tr>
</tbody>
</table>

11.45.4 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 (424) property. The master dataset must be set through the DataSource (373) property, and is usually set by the application programmer.

See also: TDetailDataLink.DetailDataset (424), TDatalink.Datasource (373)

11.45.5 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 (482)

11.45.6 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 (373)) on which the master-detail relationship is based. Setting this property will fill the TMasterDataLink.Fields (483) property with the field instances of the master dataset.

See also: TMasterDataLink.Fields (483), TDatalink.Dataset (373), TDataset.GetFieldList (388)
11.45.7 **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 (424) instances from the master dataset (TDataLink.Dataset (373)) when the FieldNames (482) property is set, and when the master dataset opens.

**See also:** TField (424), TMasterDataLink.FieldNames (482)

11.45.8 **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 (483) list is empty, TMasterDataLink.OnMasterDisable (483) is called instead.

**See also:** TMasterDataLink.OnMasterDisable (483)

11.45.9 **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 (483)

11.46 **TMasterParamsDataLink**

11.46.1 **Description**

TMasterParamsDataLink is a TDataLink (370) 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 (370), TDataSet (375), TParams (501), TParam (488)
### 11.46.2 Method overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>484</td>
<td>CopyParamsFromMaster</td>
<td>Copy parameter values from master dataset.</td>
</tr>
<tr>
<td>484</td>
<td>Create</td>
<td>Initialize a new TMasterParamsDataLink instance</td>
</tr>
<tr>
<td>484</td>
<td>RefreshParamNames</td>
<td>Refresh the list of parameter names</td>
</tr>
</tbody>
</table>

### 11.46.3 Property overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Properties</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>485</td>
<td>Params</td>
<td>rw</td>
<td>Parameters of detail dataset.</td>
</tr>
</tbody>
</table>

### 11.46.4 TMasterParamsDataLink.Create

**Synopsis:** Initialize a new TMasterParamsDataLink instance

**Declaration:**

```plaintext
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 (501) in the published properties of ADataSet and assigns it to the Params (485) property.

See also: TDataset (375), TParams (501), TMasterParamsDataLink.Params (485)

### 11.46.5 TMasterParamsDataLink.RefreshParamNames

**Synopsis:** Refresh the list of parameter names

**Declaration:**

```plaintext
procedure RefreshParamNames; Virtual
```

**Visibility:** public

**Description:** RefreshParamNames scans the Params (485) property and sets the FieldNames (482) property to the list of parameter names.

See also: TMasterParamsDataLink.Params (485), TMasterDataLink.FieldNames (482)

### 11.46.6 TMasterParamsDataLink.CopyParamsFromMaster

**Synopsis:** Copy parameter values from master dataset.

**Declaration:**

```plaintext
procedure CopyParamsFromMaster(CopyBound: Boolean); Virtual
```

**Visibility:** public

**Description:** CopyParamsFromMaster calls TParams.CopyParamValuesFromDataset (506), 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 (498) 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 (506), TParam.Bound (498)
11.46.7  **TMasterParamsDataLink.Params**

**Synopsis:** Parameters of detail dataset.

**Declaration:**

```plaintext
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 (501), TMasterParamsDataLink.Create (484)

11.47  **TMemoField**

11.47.1 **Description**

`TMemoField` is the class used when a dataset must manage memo (Text BLOB) data. (`TField.DataType (438)` equals `ftMemo`). It initializes some of the properties of the `TField (424)` class. All methods to be able to work with memo fields have been implemented in the `TBLOBField (351)` 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 (375), TField (424), TBLOBField (351), TWideMemoField (513), TGraphicField (469)

11.47.2 **Method overview**

<table>
<thead>
<tr>
<th>Page</th>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>485</td>
<td>Create</td>
<td>Create a new instance of the <code>TMemoField</code> class</td>
</tr>
</tbody>
</table>

11.47.3 **Property overview**

<table>
<thead>
<tr>
<th>Page</th>
<th>Properties</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>486</td>
<td>CodePage</td>
<td>r</td>
<td>Codepage of the memo field string data</td>
</tr>
<tr>
<td>486</td>
<td>Transliterate</td>
<td></td>
<td>Should the contents of the field be transliterated</td>
</tr>
</tbody>
</table>

11.47.4  **TMemoField.Create**

**Synopsis:** Create a new instance of the `TMemoField` class

**Declaration:**

```plaintext
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 (424)` properties to configure the instance for working with memo values.

**See also:** TField (424)
11.47.5 TMemoField.CodePage

Synopsis: Codepage of the memo field string data


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 (434), TFieldDef.CodePage (453)

11.47.6 TMemoField.Transliterate

Synopsis: Should the contents of the field be transliterated

Declaration: Property Transliterate :

Visibility: published
Access:

Description: Transliterate is redefined from TBlobField.Transliterate (355) with a default value of true.

See also: TBlobField.Transliterate (355), TStringField.Transliterate (510), TDataset.Translate (394)

11.48 TNamedItem

11.48.1 Description

NamedItem is a TCollectionItem (???) descendent which introduces a Name (487) property. It automatically returns the value of the Name property as the value of the DisplayName (486) property.

See also: DisplayName (486), Name (487)

11.48.2 Property overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Properties</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>486</td>
<td>DisplayName</td>
<td>rw</td>
<td>Display name</td>
</tr>
<tr>
<td>487</td>
<td>Name</td>
<td>rw</td>
<td>Name of the item</td>
</tr>
</tbody>
</table>

11.48.3 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 (487) property.

See also: Name (487)
### 11.48.4  TNamedItem.Name

**Synopsis:** Name of the item

**Declaration:**
```pascal
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 (486) property. If the TNamedItem item is owned by a TDefCollection (422) collection, then the name must be unique, i.e. each Name value may appear only once in the collection.

**See also:** DisplayName (486), TDefCollection (422)

---

### 11.49  TNumericField

#### 11.49.1  Description

TNumericField is an abstract class which overrides some of the methods of TField (424) 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 (488) and TNumericField.EditFormat (488).

Since TNumericField is an abstract class, it must never be instantiated directly. Instead one of the descendent classes should be created.

**See also:** TField (424), TNumericField.DisplayFormat (488), TNumericField.EditFormat (488), TField.Alignment (443), TIntegerField (475), TLargeIntField (475), TFloatField (464), TBCDField (347)

#### 11.49.2  Method overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>487</td>
<td>Create</td>
<td>Create a new instance of TNumericField</td>
</tr>
</tbody>
</table>

#### 11.49.3  Property overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Properties</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>488</td>
<td>Alignment</td>
<td></td>
<td>Alignment of the field</td>
</tr>
<tr>
<td>488</td>
<td>DisplayFormat</td>
<td>rw</td>
<td>Format string for display of numerical data</td>
</tr>
<tr>
<td>488</td>
<td>EditFormat</td>
<td>rw</td>
<td>Format string for editing of numerical data</td>
</tr>
</tbody>
</table>

#### 11.49.4  TNumericField.Create

**Synopsis:** Create a new instance of TNumericField

**Declaration:**
```pascal
constructor Create(AOwner: TComponent); Override
```

**Visibility:** public

**Description:**
Create calls the inherited constructor and then initializes the TField.Alignment (443) property with

**See also:** TField.Alignment (443)
11.49.5 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 (443)

11.49.6 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 (438) 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 (438), TNumericField.EditFormat (488)

11.49.7 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 (441) 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 (441), TNumericField.DisplayFormat (488)

11.50 TParam

11.50.1 Description

TParam is one item in a TParams (501) collection. It describes the name (TParam.Name (500)), type (ParamType (500)) and value (TParam.Value (501)) 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 (375) descendent that owns the parameters should have created all necessary TParam instances.

See also: TParams (501)
### 11.50.2 Method overview

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Assign</td>
<td>Assign one parameter instance to another</td>
</tr>
<tr>
<td>AssignField</td>
<td>Copy value from field instance</td>
</tr>
<tr>
<td>AssignFieldValue</td>
<td>Assign field value to the parameter.</td>
</tr>
<tr>
<td>AssignFromField</td>
<td>Copy field type and value</td>
</tr>
<tr>
<td>AssignToField</td>
<td>Assign parameter value to field</td>
</tr>
<tr>
<td>Clear</td>
<td>Create a new parameter value</td>
</tr>
<tr>
<td>GetData</td>
<td>Get the parameter value from a memory buffer</td>
</tr>
<tr>
<td>GetDataSize</td>
<td>Return the size of the data.</td>
</tr>
<tr>
<td>LoadFromFile</td>
<td>Load a parameter value from file</td>
</tr>
<tr>
<td>LoadFromStream</td>
<td>Load a parameter value from stream</td>
</tr>
<tr>
<td>SetBlobData</td>
<td>Set BLOB data</td>
</tr>
<tr>
<td>SetData</td>
<td>Set the parameter value from a buffer</td>
</tr>
</tbody>
</table>

### 11.50.3 Property overview

<table>
<thead>
<tr>
<th>Properties</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AsAnsiString</td>
<td>rw</td>
<td>Parameter contents as an ANSI string</td>
</tr>
<tr>
<td>AsBCD</td>
<td>rw</td>
<td>Get or set parameter value as BCD value</td>
</tr>
<tr>
<td>AsBlob</td>
<td>rw</td>
<td>Return parameter value as a blob</td>
</tr>
<tr>
<td>AsBoolean</td>
<td>rw</td>
<td>Get/Set parameter value as a boolean value</td>
</tr>
<tr>
<td>AsBytes</td>
<td>rw</td>
<td>Get or set parameter value as TBytes</td>
</tr>
<tr>
<td>AsCurrency</td>
<td>rw</td>
<td>Get/Set parameter value as a currency value</td>
</tr>
<tr>
<td>AsDate</td>
<td>rw</td>
<td>Get/Set parameter value as a date (TDateTime) value</td>
</tr>
<tr>
<td>AsDateTime</td>
<td>rw</td>
<td>Get/Set parameter value as a date/time (TDateTime) value</td>
</tr>
<tr>
<td>AsFloat</td>
<td>rw</td>
<td>Get/Set parameter value as a floating-point value</td>
</tr>
<tr>
<td>AsFMTBCD</td>
<td>rw</td>
<td>Parameter value as a BCD value</td>
</tr>
<tr>
<td>AsInteger</td>
<td>rw</td>
<td>Get/Set parameter value as an integer (32-bit) value</td>
</tr>
<tr>
<td>AsLargeInt</td>
<td>rw</td>
<td>Get/Set parameter value as a 64-bit integer value</td>
</tr>
<tr>
<td>AsMemo</td>
<td>rw</td>
<td>Get/Set parameter value as a memo (string) value</td>
</tr>
<tr>
<td>AsSmallInt</td>
<td>rw</td>
<td>Get/Set parameter value as a smallint value</td>
</tr>
<tr>
<td>AsString</td>
<td>rw</td>
<td>Get/Set parameter value as a string</td>
</tr>
<tr>
<td>AsTime</td>
<td>rw</td>
<td>Get/Set parameter value as a time (TDateTime) value</td>
</tr>
<tr>
<td>AsUnicodeString</td>
<td>rw</td>
<td>Parameter contents as a Unicode string</td>
</tr>
<tr>
<td>AsUTF8String</td>
<td>rw</td>
<td>Parameter contents as an UTF8 string</td>
</tr>
<tr>
<td>AsWideString</td>
<td>rw</td>
<td>Get/Set the value as a widestring</td>
</tr>
<tr>
<td>AsWord</td>
<td>rw</td>
<td>Get/Set parameter value as a word value</td>
</tr>
<tr>
<td>Bound</td>
<td>rw</td>
<td>Is the parameter value bound (set to fixed value)</td>
</tr>
<tr>
<td>Dataset</td>
<td>r</td>
<td>Dataset to which this parameter belongs</td>
</tr>
<tr>
<td>DataType</td>
<td>rw</td>
<td>Data type of the parameter</td>
</tr>
<tr>
<td>IsNull</td>
<td>r</td>
<td>Is the parameter empty</td>
</tr>
<tr>
<td>Name</td>
<td>rw</td>
<td>Name of the parameter</td>
</tr>
<tr>
<td>NativeStr</td>
<td>rw</td>
<td>No description available</td>
</tr>
<tr>
<td>NumericScale</td>
<td>rw</td>
<td>Numeric scale</td>
</tr>
<tr>
<td>ParamType</td>
<td>rw</td>
<td>Type of parameter</td>
</tr>
<tr>
<td>Precision</td>
<td>rw</td>
<td>Precision of the BCD value</td>
</tr>
<tr>
<td>Size</td>
<td>rw</td>
<td>Size of the parameter</td>
</tr>
<tr>
<td>Text</td>
<td>rw</td>
<td>Read or write the value of the parameter as a string</td>
</tr>
<tr>
<td>Value</td>
<td>rws</td>
<td>Value as a variant</td>
</tr>
</tbody>
</table>
11.50.4 TParam.Create

Synopsis: Create a new parameter value

Declaration: constructor Create(ACollection: TCollection); Override; Overload
constructor Create(AParams: TParams; AParmType: 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 (AParmType), in which case the owning TParams collection must be specified in AParams

See also: TParams (501)

11.50.5 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 (424), then it is passed to TParam.AssignField (490). If Source is of type TStrings, then it is assigned to TParams.AsMemo (501).

Errors: If Source is not of type TParam, TField or TStrings, an exception will be raised.

See also: TField (424), TParam.Name (500), TParam.Bound (498), TParam.NumericScale (500), TParam.ParamType (500), TParam.value (501), TParam.Size (501), TParam.AssignField (490), Tparam.AsMemo (496)

11.50.6 TParam.AssignField

Synopsis: Copy value from field instance

Declaration: procedure AssignField(Field: TField)

Visibility: public

Description: AssignField copies the Field, FieldName (445) and Value (442) 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 (490), TParam.AssignToField (490), TParam.AssignFieldValue (491)

11.50.7 TParam.AssignToField

Synopsis: Assign parameter value to field

Declaration: procedure AssignToField(Field: TField)

Visibility: public

Description: AssignToField copies the parameter value (501) to the field instance. If Field is Nil, nothing happens.
CHAPTER 11. REFERENCE FOR UNIT 'DB'

Errors: An EDatabaseError (338) exception is raised if the field has an unsupported field type (for types ftCursor, ftArray, ftDataset, ftReference).

See also: TParam.Assign (490), TParam.AssignField (490), TParam.AssignFromField (491)

11.50.8 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 (498) bound parameter to True. This method is called from TParam.AssignField (490).

See also: TField (424), TParam.AssignField (490), TParam.Bound (498)

11.50.9 TParam.AssignFromField

Synopsis: Copy field type and value

Declaration: procedure AssignFromField(Field: TField)

Visibility: public

Description: AssignFromField copies the field value (442) and data type (TField.DataType (438)) to the parameter instance. If Field is Nil, nothing happens. This is the reverse operation of TParam.AssignToField (490).

Errors: An EDatabaseError (338) exception is raised if the field has an unsupported field type (for types ftCursor, ftArray, ftDataset, ftReference).

See also: TParam.Assign (490), TParam.AssignField (490), TParam.AssignToField (490)

11.50.10 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 (501), TParam.Name (500), TParam.ParamType (500), TParam.DataType (499)

11.50.11 TParam.GetData

Synopsis: Get the parameter value from a memory buffer

Declaration: procedure GetData(Buffer: Pointer)

Visibility: public
CHAPTER 11. REFERENCE FOR UNIT 'DB'

Description: GetData retrieves the parameter value and stores it in buffer. It uses the same data layout as TField (424), 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 (338) exception.

See also: TParam.SetData (493), TField (424)

11.50.12 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 (338) exception is raised when this function is called.

See also: TParam.GetData (491), TParam.SetData (493)

11.50.13 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 (492).

Errors: If the specified FileName is not a valid file, or the file is not readable, an exception will occur.

See also: TParam.LoadFromStream (492), TBlobType (322), TParam.SaveToFile (488)

11.50.14 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 (492), TParam.SaveToStream (488)
11.50.15 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 (492)

11.50.16 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 (491): it reads the parameter value from the memory area pointed to by Buffer. The size of the data read is determined by TParam.GetDataSize (492) and the type of data by TParam.DataType (499) : it is the same storage mechanism used by TField (424), and so can be used to copy the value from a TDataset (375) record buffer.

Errors: Not all field types are supported. If an unsupported field type is encountered, an EDatabaseError (338) exception is raised.

See also: TDataset (375), TParam.GetData (491), TParam.DataType (499), TParam.GetDataSize (492)

11.50.17 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 (495)

11.50.18 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 (496)

493
11.50.19 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 (499), TParam.Value (501)

11.50.20 TParam.AsBytes

Synopsis: Get or set parameter value as TBytes

Declaration: Property AsBytes : TBytes

Visibility: public

Access: Read, Write

Description: AsBytes 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: TParamAsString (496)

11.50.21 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 (495), TParam.DataType (499), TParam.Value (501)

11.50.22 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 (495), TParam.AsTime (497), TParam.DataType (499), TParam.Value (501)
11.50.23  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 (494), TParam.asTime (497), TParam.DataType (499), TParam.Value (501)

11.50.24  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 (494), TParam.DataType (499), TParam.Value (501)

11.50.25  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 (495), TParam.AsSmallInt (496), TParam.AsWord (497), TParam.DataType (499), TParam.Value (501)

11.50.26  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 (495), TParam.asSmallint (496), TParam.AsWord (497), TParam.DataType (499), TParam.Value (501)
11.50.27 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 (496), TParam.LoadFromStream (492), TParam.SaveToStream (488), TParam.DataType (499), TParam.Value (501)

11.50.28 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 (495), TParam.AsLargeInt (495), TParam.AsWord (497), TParam.DataType (499), TParam.Value (501)

11.50.29 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 (499), TParam.Value (501)

11.50.30 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 (496), TParam.AsUnicodeString (497), TParam.AsUTF8String (497)
11.50.31 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 (496), TParam.AsUnicodeString (497), TParam.AsAnsiString (496)

11.50.32 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 (496), TParam.AsUTF8String (497), TParam.AsAnsiString (496)

11.50.33 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 (494), TParam.AsDateTime (495), TParam.DataType (499), TParam.Value (501)

11.50.34 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 (495), TParam.AsLargeInt (495), TParam.AsSmallInt (496), TParam.DataType (499), TParam.Value (501)
11.50.35  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 (319), AsCurrency (319)

11.50.36  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 (499), TParam.Value (501)

11.50.37  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 (501) 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 (375), TParams (501)

11.50.38  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 (491), TParam.Value (501)
11.50.39   TParam.NativeStr
Synopsis: No description available

Declaration: Property NativeStr : string
Visibility: public
Access: Read, Write

Description: No description available

11.50.40   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 (496), but, when written, does not set the data type; instead, it attempts to convert the value to the type specified in TParam.DataType (499).

See also: TParam.AsString (496), TParam.DataType (499)

11.50.41   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 (499) to ftWideString.

See also: TParam.AsString (496), TParam.Value (501), TParam.DataType (499)

11.50.42   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 (498), TParam.Value (501), TParam.AssignField (490)
11.50.43 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 (499), TParam.Value (501), TParams.ParamByName (504)

11.50.44 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 (501), TParam.Size (501)

11.50.45 TParam.ParamType

Synopsis: Type of parameter

Declaration: Property ParamType : TParamType

Visibility: published

Access: Read, Write

Description: ParamType 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:

- ptUnknown Unknown type
- ptInput Input parameter
- ptOutput Output parameter, filled on result
- ptInputOutput Input/output parameter
- ptResult Result parameter

The ParamType property is usually set by the database engine that creates the parameter instances.

See also: TParam.DataType (499), TParam.DataSize (488), TParam.Name (500)
11.50.46  **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 (500), TParam.Size (501)

11.50.47  **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 (438) in the TParam.AssignFieldValue (491) method. The actual size can be retrieved through the TParam.Datasize (488) property.

See also: TParam.Datasize (488), TField.DataSize (438), TParam.AssignFieldValue (491)

11.50.48  **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 (499)

11.51  **TParams**

11.51.1  **Description**

TParams is a collection of TParam (488) 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 (488) 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 (504).

TDataset (375) 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 (483) 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 (506) method exists.
CHAPTER 11. REFERENCE FOR UNIT 'DB'

See also: TDataset (375), TMasterParamsDataLink (483), TParam (488), TParams.ParseSQL (504), TParams.CopyParamValuesFromDataset (506)

11.51.2 Method overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>502</td>
<td>AddParam</td>
<td>Add a parameter to the collection</td>
</tr>
<tr>
<td>503</td>
<td>AssignValues</td>
<td>Copy values from another collection</td>
</tr>
<tr>
<td>506</td>
<td>CopyParamValuesFromDataset</td>
<td>Copy parameter values from the fields in a dataset.</td>
</tr>
<tr>
<td>502</td>
<td>Create</td>
<td>Create a new instance of TParams</td>
</tr>
<tr>
<td>503</td>
<td>CreateParam</td>
<td>Create and add a new parameter to the collection</td>
</tr>
<tr>
<td>503</td>
<td>FindParam</td>
<td>Find a parameter with given name</td>
</tr>
<tr>
<td>504</td>
<td>GetEnumerator</td>
<td>Return an enumerator for the parameters</td>
</tr>
<tr>
<td>503</td>
<td>GetParamList</td>
<td>Fetch a list of TParam instances</td>
</tr>
<tr>
<td>504</td>
<td>IsEqual</td>
<td>Is the list of parameters equal</td>
</tr>
<tr>
<td>504</td>
<td>ParamByName</td>
<td>Return a parameter by name</td>
</tr>
<tr>
<td>504</td>
<td>ParseSQL</td>
<td>Parse SQL statement, replacing parameter names with SQL parameter placeholders</td>
</tr>
<tr>
<td>505</td>
<td>RemoveParam</td>
<td>Remove a parameter from the collection</td>
</tr>
</tbody>
</table>

11.51.3 Property overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Properties</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>506</td>
<td>Dataset</td>
<td>r</td>
<td>Dataset that owns the TParams instance</td>
</tr>
<tr>
<td>506</td>
<td>Items</td>
<td>rw</td>
<td>Indexed access to TParams instances in the collection</td>
</tr>
<tr>
<td>506</td>
<td>ParamValues</td>
<td>rw</td>
<td>Named access to the parameter values.</td>
</tr>
</tbody>
</table>

11.51.4 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 (488) 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 (488)

11.51.5 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.
11.51.6 TParams.AssignValues

Synopsis: Copy values from another collection

Declaration: procedure AssignValues(Value: TParams)

Visibility: public

Description: AssignValues examines all TParam (488) 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 (490)). If it is not found, nothing is done.

See also: TParam (488), TParam.Assign (490)

11.51.7 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 (488) 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 (488), TParam.Name (500), TParam.Datatype (499), TParam.Paramtype (500)

11.51.8 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 (488) 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 (500), TParams.ParamByName (504), TParams.GetParamList (503)

11.51.9 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 (488), TParam.Name (500), TParams.ParamByName (504)`

### 11.51.10 TParams.IsEqual

**Synopsis:** Is the list of parameters equal

**Declaration:**

```pascal
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 (488)`

### 11.51.11 TParams.GetEnumerator

**Synopsis:** Return an enumerator for the parameters

**Declaration:**

```pascal
function GetEnumerator : TParamsEnumerator
```

**Visibility:** public

**Description:**

`GetEnumerator` returns an enumerator that loops over all parameters (as implemented by `TParamsEnumerator (507)`)

See also: `TPParamsEnumerator (507)`

### 11.51.12 TParams.ParamByName

**Synopsis:** Return a parameter by name

**Declaration:**

```pascal
function ParamByName(const Value: string) : TParam
```

**Visibility:** public

**Description:**

`ParamByName` searches the collection for the `TParam (488)` 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 (500), TParams.FindParam (503), TParams.GetParamList (503)`

### 11.51.13 TParams.ParseSQL

**Synopsis:** Parse SQL statement, replacing parameter names with SQL parameter placeholders

**Declaration:**

```pascal
function ParseSQL(SQL: string; DoCreate: Boolean) : string; Overload
function ParseSQL(SQL: string; DoCreate: Boolean; EscapeSlash: Boolean; EscapeRepeat: Boolean; ParameterStyle: TParamStyle) : string; Overload
```

504
function ParseSQL(SQL: string; DoCreate: Boolean; EscapeSlash: Boolean;
    EscapeRepeat: Boolean; ParameterStyle: TParamStyle;
    out ParamBinding: TParamBinding) : string; Overload
function ParseSQL(SQL: string; DoCreate: Boolean; EscapeSlash: Boolean;
    EscapeRepeat: Boolean; ParameterStyle: TParamStyle;
    out ParamBinding: TParamBinding;
    out ReplaceString: string) : string; Overload
function ParseSQL(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:

- **psInterbase**: Parameters are specified by a ? character
- **psPostgreSQL**: Parameters are specified by a $N character.
- **psSimulated**: Parameters 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 (488), TParam.Name (500), TParamStyle (331)

11.51.14 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 (488)
11.51.15  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 (490). By default, this operation is only performed on TParam instances with their Bound (498) 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 (488), TParam.Bound (498), TParam.AssignField (490), TDataset (375), TDataset.FieldByName (385)

11.51.16  TParams.Dataset

Synopsis: Dataset that owns the TParams instance

Declaration: Property Dataset : TDataSet

Visibility: public
Access: Read

Description: Dataset is the TDataset (375) instance that was specified when the TParams instance was created.

See also: TParams.Create (502), TDataset (375)

11.51.17  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 (501)

11.51.18  TParams.ParamValues

Synopsis: Named access to the parameter values.

Declaration: Property ParamValues[ParamName: string]: Variant

Visibility: public
11.52 TParamsEnumerator

11.52.1 Description

TParamsEnumerator is a helper class to implement enumeration (for..in) of parameters. It implements the IEnumerator interface.

See also: TParams.GetEnumerator (504)

11.52.2 Method overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>507</td>
<td>Create</td>
<td>Create a new TParamsEnumerator instance</td>
</tr>
<tr>
<td>507</td>
<td>MoveNext</td>
<td>Go to next TParam</td>
</tr>
</tbody>
</table>

11.52.3 Property overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Properties</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>508</td>
<td>Current</td>
<td>r</td>
<td>Current TParam instance</td>
</tr>
</tbody>
</table>

11.52.4 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 (504)

11.52.5 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 (508) will return the new current TParam

See also: TParamsEnumerator.Current (508)
11.52.6  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 (507) returned true.

See also: TParamsEnumerator.MoveNext (507)

11.53  TSmallintField

11.53.1  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 (424), TNumericField (487), TLongintField (477), TWordField (515)

11.53.2  Method overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>508</td>
<td>Create</td>
<td>Create a new instance of the TSmallintField class.</td>
</tr>
</tbody>
</table>

11.53.3  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 (508) class. It calls the inherited constructor and then simply sets some of the TField (424) properties to work with 16-bit signed integer data.

See also: TField (424)

11.54  TStringField

11.54.1  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 (424) 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 (424), TWideStringField (514), TDataset (375)

### 11.54.2 Method overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>509</td>
<td>Create</td>
<td>Create a new instance of the TStringField class</td>
</tr>
<tr>
<td>509</td>
<td>SetFieldType</td>
<td>Set the field type</td>
</tr>
</tbody>
</table>

### 11.54.3 Property overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Properties</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>509</td>
<td>CodePage</td>
<td>r</td>
<td>Codepage of the field string data</td>
</tr>
<tr>
<td>510</td>
<td>EditMask</td>
<td></td>
<td>Specify an edit mask for an edit control</td>
</tr>
<tr>
<td>510</td>
<td>FixedChar</td>
<td>rw</td>
<td>Is the string declared with a fixed length ?</td>
</tr>
<tr>
<td>511</td>
<td>Size</td>
<td></td>
<td>Maximum size of the string</td>
</tr>
<tr>
<td>510</td>
<td>Transliterate</td>
<td>rw</td>
<td>Should the field value be transliterated when reading or writing</td>
</tr>
<tr>
<td>510</td>
<td>Value</td>
<td>rw</td>
<td>Value of the field as a string</td>
</tr>
</tbody>
</table>

### 11.54.4 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 (424) properties after having called the inherited constructor.

### 11.54.5 TStringField.SetFieldType

Synopsis: Set the field type

Declaration: `procedure SetFieldType(AValue: TFieldType); Override`

Visibility: public

Description: `SetFieldType` is overridden in TStringField (508) to check the data type more accurately (ftString and ftFixedChar). No extra functionality is added.

See also: TField.DataType (438)

### 11.54.6 TStringField.CodePage

Synopsis: Codepage of the field string data


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 (434), TField.AsUnicodeString (435), TField.AsAnsi8String (424), TFieldDef.CodePage (453)
11.54.7  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.

11.54.8  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 (394).

See also: TDataset.Translate (394)

11.54.9  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 (434) when read, or sets theAsString property when written to.

See also: TField.AsString (434), TField.Value (442)

11.54.10  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 (439)

11.54.11 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 (441)

11.55 TTimeField

11.55.1 Description

TTimeField is the class used when a dataset must manage data of type time. (TField.DataType
(438) equals ftTime). It initializes some of the properties of the TField (424) 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 (375), TField (424), TDateTimeField (417), TDateField (416)

11.55.2 Method overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>511</td>
<td>Create</td>
<td>Create a new instance of a TTimeField class.</td>
</tr>
</tbody>
</table>

11.55.3 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 (424) properties to configure the instance for working with time values.

See also: TField (424)
11.56  TVarBytesField

11.56.1  Description
TVarBytesField is the class used when a dataset must manage data of variable-size binary type. (TField.DataType (438) equals ftVarBytes). It initializes some of the properties of the TField (424) 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 (375), TField (424), TBytesField (357)

11.56.2  Method overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>512</td>
<td>Create</td>
<td>Create a new instance of a TVarBytesField class.</td>
</tr>
</tbody>
</table>

11.56.3  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 (424) properties to configure the instance for working with variable-size binary data values.

See also: TField (424)

11.57  TVariantField

11.57.1  Description
TVariantField is the class used when a dataset must manage native variant-typed data. (TField.DataType (438) equals ftVariant). It initializes some of the properties of the TField (424) 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 (375), TField (424)

11.57.2  Method overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>513</td>
<td>Create</td>
<td>Create a new instance of the TVariantField class</td>
</tr>
</tbody>
</table>
11.57.3 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 (424) properties to configure the instance for working with variant values.

See also: TField (424)

11.58 TWideMemoField

11.58.1 Description

TWideMemoField is the class used when a dataset must manage memo (Text BLOB) data. (TField.DataType (438) equals ftWideMemo). It initializes some of the properties of the TField (424) class. All methods to be able to work with widestring memo fields have been implemented in the TBlobField (351) 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 (375), TField (424), TBLOBField (351), TMemoField (485), TGraphicField (469)

11.58.2 Method overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>513</td>
<td>Create</td>
<td>Create a new instance of the TWideMemoField class</td>
</tr>
</tbody>
</table>

11.58.3 Property overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Properties</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>514</td>
<td>Value</td>
<td>rw</td>
<td>Value of the field’s contents as a widestring</td>
</tr>
</tbody>
</table>

11.58.4 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 (424) properties to configure the instance for working with widestring memo values.

See also: TField (424)
11.58.5 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 (436) property.

See also: TField.Value (442), TField.AsWideString (436)

11.59 TWideStringField

11.59.1 Description

TWideStringField is the string field class instantiated for fields of data type ftWideString. This class overrides some of the standard TField (424) 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 (424), TStringField (508), TDataset (375)

11.59.2 Method overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>514</td>
<td>Create</td>
<td>Create a new instance of the TWideStringField class.</td>
</tr>
<tr>
<td>514</td>
<td>SetFieldType</td>
<td>Set the field type</td>
</tr>
</tbody>
</table>

11.59.3 Property overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Properties</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>515</td>
<td>Value</td>
<td>rw</td>
<td>Value of the field as a widestring</td>
</tr>
</tbody>
</table>

11.59.4 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 (424) properties after having called the inherited constructor.

11.59.5 TWideStringField.SetFieldType

Synopsis: Set the field type

Declaration: procedure SetFieldType(AValue: TFieldType); Override
Visibility: public

Description: SetFieldType is overridden in TWideStringField (514) to check the data type more accurately (ftWideString and ftFixedWideChar). No extra functionality is added.

See also: TField.DataType (438)

11.59.6 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 (436) property.

See also: TField.AsWideString (436), TField.Value (442)

11.60 TWordField

11.60.1 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 (424), TNumericField (487), TLongintField (477), TSmallIntField (508)

11.60.2 Method overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>515</td>
<td>Create</td>
<td>Create a new instance of the TWordField class.</td>
</tr>
</tbody>
</table>

11.60.3 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 (515) class. It calls the inherited constructor and then simply sets some of the TField (424) properties to work with 16-bit unsigned integer data.

See also: TField (424)
Chapter 12

Reference for unit ’dbugintf’

12.1 Used units

Table 12.1: Used units by unit ’dbugintf’

<table>
<thead>
<tr>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>dbugmsg</td>
<td>523</td>
</tr>
<tr>
<td>System</td>
<td>??</td>
</tr>
</tbody>
</table>

12.2 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 (516) 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 (516) unit should also be functional.

12.3 Writing a debug server

Writing a debug server is relatively easy. It should instantiate a TSimpleIPCServer class from the SimpleIPC (516) 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 (516) 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 (516) unit should also be functional.

12.4 Constants, types and variables

12.4.1 Resource strings

SEntering = '> Entering '
String used when sending method enter message.

SExitng = '< 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

12.4.2 Types

TDebugLevel = (dlInformation,dlWarning,dlError)

<table>
<thead>
<tr>
<th>Value</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>dlError</td>
<td>Error message</td>
</tr>
<tr>
<td>dlInformation</td>
<td>Informational message</td>
</tr>
<tr>
<td>dlWarning</td>
<td>Warning message</td>
</tr>
</tbody>
</table>

TDebugLevel indicates the severity level of the debug message to be sent. By default, an informational message is sent.

TErrorLevel = Array[TDebugLevel] of Integer

12.4.3 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.

### 12.5 Procedures and functions

#### 12.5.1 FreeDebugClient

**Synopsis:**

**Declaration:** procedure FreeDebugClient

**Visibility:** default

**Description:**

#### 12.5.2 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 (522), SendDebug (519)

#### 12.5.3 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 (519) call will initialize the debug client when it is first called.

**Errors:** None.

See also: SendDebug (519), StartDebugServer (522)
12.5.4 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 (519) which sends the name and value of a boolean value as an informational message.

Errors: None.

See also: SendDebug (519), SendDateTime (519), SendInteger (520), SendPointer (521)

12.5.5 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 (519) 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 (519), SendBoolean (519), SendInteger (520), SendPointer (521)

12.5.6 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 (518).

See also: SendDebugEx (519), SendDebugFmt (520), SendDebugFmtEx (520)

12.5.7 SendDebugEx

Synopsis: Send debug message other than informational messages

Declaration: procedure SendDebugEx(const Msg: string; MType: TDebugLevel)

Visibility: default
**CHAPTER 12. REFERENCE FOR UNIT 'DBGINTF'**

**Description:** `SendDebugEx` allows to specify the debug level of the message to be sent in `MType`. By default, `SendDebug (519)` uses informational messages. Other than that the function of `SendDebugEx` is equal to that of `SendDebug`.

**Errors:** None.

See also: `SendDebug (519), SendDebugFmt (520), SendDebugFmtEx (520)`

---

### 12.5.8 SendDebugFmt

**Synopsis:** Format and send a debug message

**Declaration:**
```pascal
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 (519)`. It exists mainly to avoid the `Format` call in calling code.

**Errors:** None.

See also: `SendDebug (519), SendDebugEx (519), SendDebugFmtEx (520), #rtl.sysutils.format (??)`

---

### 12.5.9 SendDebugFmtEx

**Synopsis:** Format and send message with alternate type

**Declaration:**
```pascal
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 (519)` with Debug level `MType`. It exists mainly to avoid the `Format` call in calling code.

**Errors:** None.

See also: `SendDebug (519), SendDebugEx (519), SendDebugFmtEx (520), #rtl.sysutils.format (??)`

---

### 12.5.10 SendInteger

**Synopsis:** Send the value of an integer variable.

**Declaration:**
```pascal
procedure SendInteger(const Identifier: string; const Value: Integer; HexNotation: Boolean)
```

**Visibility:** default

**Description:** `SendInteger` is a simple wrapper around `SendDebug (519)` 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 (519), SendBoolean (519), SendDateTime (519), SendPointer (521)`
12.5.11 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 (521), 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 (519), SendMethodExit (521), SendSeparator (522)

12.5.12 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 (521), 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 (519), SendMethodEnter (521), SendSeparator (522)

12.5.13 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 (519) 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 (519), SendBoolean (519), SendDateTime (519), SendInteger (520)
CHAPTER 12. REFERENCE FOR UNIT ‘DEBUGINTF’

12.5.14  SendSeparator

Synopsis: Send a separator message

Declaration: procedure SendSeparator

Visibility: default

Description: SendSeparator is a simple wrapper around SendDebug (519) 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 (519), SendMethodEnter (521), SendMethodExit (521)

12.5.15  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 (519) 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 (518), SendDebug (519)

12.5.16  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 (519) call will attempt to start the server by itself if it is not yet running.

Errors: On error, False is returned.

See also: SendDebug (519), InitDebugClient (518)
Chapter 13

Reference for unit ’dbugmsg’

13.1 Used units

<table>
<thead>
<tr>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Classes</td>
<td>??</td>
</tr>
</tbody>
</table>

13.2 Overview

dbugmsg is an auxiliary unit used in the dbugintf (516) unit. It defines the message protocol used between the debug unit and the debug server.

13.3 Constants, types and variables

13.3.1 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.

### 13.3.2 Types

### 13.4 Procedures and functions

#### 13.4.1 DebugMessageName

**Synopsis:** Return the name of the debug message

**Declaration:**

```pascal
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 (525) 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.

#### 13.4.2 ReadDebugMessageFromStream

**Synopsis:** Read a message from stream

**Declaration:**

```pascal
procedure ReadDebugMessageFromStream(AStream: TStream; 
    var Msg: TDebugMessage)
```

**Visibility:** default

**Description:** ReadDebugMessageFromStream reads a TDebugMessage (525) 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 (525), WriteDebugMessageToStream (524)

#### 13.4.3 WriteDebugMessageToStream

**Synopsis:** Write a message to stream

**Declaration:**

```pascal
procedure WriteDebugMessageToStream(AStream: TStream; 
    const Msg: TDebugMessage)
```

**Visibility:** default
Description: WriteDebugMessageFromStream writes a TDebugMessage (525) 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 (525), ReadDebugMessageFromStream (524)

13.5 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 (524) and ReadDebugMessageFromStream (524) can be used to read or write the message from/to a stream.
Chapter 14

Reference for unit ’eventlog’

14.1 Used units

Table 14.1: Used units by unit ’eventlog’

<table>
<thead>
<tr>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Classes</td>
<td>??</td>
</tr>
<tr>
<td>System</td>
<td>??</td>
</tr>
<tr>
<td>sysutils</td>
<td>??</td>
</tr>
</tbody>
</table>

14.2 Overview

The EventLog unit implements the TEventLog (528) 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.

14.3 Constants, types and variables

14.3.1 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'

SErrLogOpenStdOut = 'Standard Output not available for logging'

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

14.3.2 Types

TLogCategoryEvent = procedure(Sender: TObject; var Code: Word) of object

TLogCategoryEvent is the event type for the TEventLog.OnGetCustomCategory (535) 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 (536) and OnGetCustomEventID (536) 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

TLogType = (ltSystem, ltFile, ltStdOut, ltStdErr)

Table 14.2: Enumeration values for type TLogType

<table>
<thead>
<tr>
<th>Value</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>ltFile</td>
<td>Write to file</td>
</tr>
<tr>
<td>ltStdErr</td>
<td></td>
</tr>
<tr>
<td>ltStdOut</td>
<td></td>
</tr>
<tr>
<td>ltSystem</td>
<td>Use the system log</td>
</tr>
</tbody>
</table>

TLogType determines where the log messages are written. It is the type of the TEventLog.LogType (533) 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.
14.4 ELogError

14.4.1 Description

ELogError is the exception used in the TEventLog (528) component to indicate errors.

See also: TEventLog (528)

14.5 TEventLog

14.5.1 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 (531) call, or the specialized Warning (531), Error (532), Info (532) or Debug (532) calls, which have the event type predefined.

See also: Log (531), Warning (531), Error (532), Info (532), Debug (532)

14.5.2 Method overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>532</td>
<td>Debug</td>
<td>Log a debug message</td>
</tr>
<tr>
<td>529</td>
<td>Destroy</td>
<td>Clean up TEventLog instance</td>
</tr>
<tr>
<td>532</td>
<td>Error</td>
<td>Log an error message to</td>
</tr>
<tr>
<td>529</td>
<td>EventTypeToString</td>
<td>Create a string representation of an event type</td>
</tr>
<tr>
<td>532</td>
<td>Info</td>
<td>Log an informational message</td>
</tr>
<tr>
<td>531</td>
<td>Log</td>
<td>Log a message to the system log.</td>
</tr>
<tr>
<td>531</td>
<td>Pause</td>
<td>Pause the sending of log messages.</td>
</tr>
<tr>
<td>529</td>
<td>RegisterMessageFile</td>
<td>Register message file</td>
</tr>
<tr>
<td>531</td>
<td>Resume</td>
<td>Resume sending of log messages if sending was paused</td>
</tr>
<tr>
<td>530</td>
<td>UnRegisterMessageFile</td>
<td>Unregister the message file (needed on windows only)</td>
</tr>
<tr>
<td>531</td>
<td>Warning</td>
<td>Log a warning message.</td>
</tr>
</tbody>
</table>
14.5.3 Property overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Properties</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>533</td>
<td>Active</td>
<td>rw</td>
<td>Activate the log mechanism</td>
</tr>
<tr>
<td>532</td>
<td>AppendContent</td>
<td>rw</td>
<td>Control whether output is appended to an existing file</td>
</tr>
<tr>
<td>535</td>
<td>CustomLogType</td>
<td>rw</td>
<td>Custom log type ID</td>
</tr>
<tr>
<td>534</td>
<td>DefaultEventType</td>
<td>rw</td>
<td>Default event type for the Log (531) call.</td>
</tr>
<tr>
<td>535</td>
<td>EventIDOffset</td>
<td>rw</td>
<td>Offset for event ID messages identifiers</td>
</tr>
<tr>
<td>534</td>
<td>FileName</td>
<td>rw</td>
<td>File name for log file</td>
</tr>
<tr>
<td>533</td>
<td>Identification</td>
<td>rw</td>
<td>Identification string for messages</td>
</tr>
<tr>
<td>533</td>
<td>LogType</td>
<td>rw</td>
<td>Log type</td>
</tr>
<tr>
<td>535</td>
<td>OnGetCustomCategory</td>
<td>rw</td>
<td>Event to retrieve custom message category</td>
</tr>
<tr>
<td>536</td>
<td>OnGetCustomEvent</td>
<td>rw</td>
<td>Event to retrieve custom event Code</td>
</tr>
<tr>
<td>536</td>
<td>OnGetCustomEventID</td>
<td>rw</td>
<td>Event to retrieve custom event ID</td>
</tr>
<tr>
<td>536</td>
<td>OnLogMessage</td>
<td>rw</td>
<td></td>
</tr>
<tr>
<td>536</td>
<td>Paused</td>
<td>rw</td>
<td>Is the message sending paused ?</td>
</tr>
<tr>
<td>534</td>
<td>RaiseExceptionOnError</td>
<td>rw</td>
<td>Determines whether logging errors are reported or ignored</td>
</tr>
<tr>
<td>534</td>
<td>TimeStampFormat</td>
<td>rw</td>
<td>Format for the timestamp string</td>
</tr>
</tbody>
</table>

14.5.4 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 (533) property to False.

See also: Active (533)

14.5.5 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 (??)

14.5.6 TEventLog.RegisterMessageFile

Synopsis: Register message file

Declaration: function RegisterMessageFile(AFileName: string) : Boolean; Virtual

Visibility: public
Description: \texttt{RegisterMessageFile} is used on Windows to register the file \texttt{AFileName} containing the formatting strings for the system messages. This should be a file containing resource strings. If \texttt{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:

\textbf{Category strings} these should be numbered from 1 to 4

1 Should contain the description of the \texttt{etInfo} event type.
2 Should contain the description of the \texttt{etWarning} event type.
4 Should contain the description of the \texttt{etError} event type.
4 Should contain the description of the \texttt{etDebug} event type.

None of these strings should have a string substitution placeholder.

The second type of strings are the \textbf{message definitions}. Their number starts at EventIDOffset (535) (default is 1000) and each string should have 1 placeholder.

Free Pascal comes with a \texttt{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 \texttt{fclel.mc} and \texttt{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.

\textbf{Errors:} If \texttt{AFileName} is invalid, false is returned.

\subsection{14.5.7 \texttt{TEventLog.UnRegisterMessageFile}}

\textbf{Synopsis:} Unregister the message file (needed on windows only)

\textbf{Declaration:} \texttt{function UnRegisterMessageFile : Boolean; Virtual}

\textbf{Visibility:} public

\textbf{Description:} \texttt{UnRegisterMessageFile} can be used to unregister a message file previously registered with \texttt{TEventLog.RegisterMessageFile} (529). 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.

\textbf{See also:} \texttt{TEventLog.RegisterMessageFile} (529)
14.5.8 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 (531).

See also: TEventLog.Resume (531), TEventLog.Paused (536)

14.5.9 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 (526).

See also: TEventLog.Pause (531), TEventLog.Paused (536)

14.5.10 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 (534).

If EventType is etCustom, then the OnGetCustomEvent (536), OnGetCustomEventID (536) and OnGetCustomCategory (535).

The other logging calls: Info (532), Warning (531), Error (532) and Debug (532) use the Log call to do the actual work.

See also: Info (532), Warning (531), Error (532), Debug (532), OnGetCustomEvent (536), OnGetCustomEventID (536), OnGetCustomCategory (535)

14.5.11 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 (531), Info (532), Error (532), Debug (532)

### 14.5.12 TEventLog.Error

**Synopsis:** Log an error message to

**Declaration:**
```delphi
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 (531), Info (532), Warning (531), Debug (532)

### 14.5.13 TEventLog.Debug

**Synopsis:** Log a debug message

**Declaration:**
```delphi
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 (531), Info (532), Warning (531), Error (532)

### 14.5.14 TEventLog.Info

**Synopsis:** Log an informational message

**Declaration:**
```delphi
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 (531), Warning (531), Error (532), Debug (532)

### 14.5.15 TEventLog.AppendContent

**Synopsis:** Control whether output is appended to an existing file

**Declaration:**
```delphi
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 (533), FileName (534)

14.5.16 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 (533), TimeStampFormat (534)

14.5.17 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 (534) property.

See also: FileName (534)

14.5.18 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 (531)
14.5.19 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.

14.5.20 TEventLog.DefaultEventType
Synopsis: Default event type for the Log (531) call.
Declaration: Property DefaultEventType : TEventType
   Visibility: published
   Access: Read, Write
Description: DefaultEventType is the event type used by the Log (531) call if it's EventType parameter is omitted.
See also: Log (531)

14.5.21 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 (533) 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 (533)

14.5.22 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 (533)
14.5.23  TEventLog.CustomLogType

Synopsis: Custom log type ID

Declaration: Property CustomLogType : Word

Visibility: published
Access: Read, Write

Description: CustomLogType is used in the EventTypeToString (529) to format the custom log event type string.

See also: EventTypeToString (529)

14.5.24  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 (529) 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 (529)

14.5.25  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 (536), OnGetCustomEvent (536)
14.5.26  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 (535), OnGetCustomEvent (536)

14.5.27  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 (535), OnGetCustomEventID (536)

14.5.28  TEventLog.OnLogMessage

Declaration: Property OnLogMessage : TLogMessageEvent

Visibility: published

Access: Read, Write

14.5.29  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 (531). Setting it to False has the same effect as calling Resume (531).

See also: TEventLog.Pause (531), TEventLog.Resume (531)
Chapter 15

Reference for unit ’ezcgi’

15.1 Used units

Table 15.1: Used units by unit ’ezcgi’

<table>
<thead>
<tr>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Classes</td>
<td>??</td>
</tr>
<tr>
<td>System</td>
<td>??</td>
</tr>
<tr>
<td>sysutils</td>
<td>??</td>
</tr>
</tbody>
</table>

15.2 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 (540) or DoGet (540) methods should be overridden.

15.3 Constants, types and variables

15.3.1 Constants

hexTable = '0123456789ABCDEF'

String constant used to convert a number to a hexadecimal code or back.

15.4 ECGIException

15.4.1 Description

Exception raised by TEZcgi (538)

See also: TEZcgi (538)
15.5 TEZcgi

15.5.1 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 (540) or DoGet (540) methods. To run the program, an instance of this class must be created, and it’s Run (539) 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.

15.5.2 Method overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>538</td>
<td>Create</td>
<td>Creates a new instance of the TEZCGI component</td>
</tr>
<tr>
<td>538</td>
<td>Destroy</td>
<td>Removes the TEZCGI component from memory</td>
</tr>
<tr>
<td>540</td>
<td>DoGet</td>
<td>Method to handle GET requests</td>
</tr>
<tr>
<td>540</td>
<td>DoPost</td>
<td>Method to handle POST requests</td>
</tr>
<tr>
<td>540</td>
<td>GetValue</td>
<td>Return the value of a request variable.</td>
</tr>
<tr>
<td>539</td>
<td>PutLine</td>
<td>Send a line of output to the web-client</td>
</tr>
<tr>
<td>539</td>
<td>Run</td>
<td>Run the CGI application.</td>
</tr>
<tr>
<td>539</td>
<td>WriteContent</td>
<td>Writes the content type to standard output</td>
</tr>
</tbody>
</table>

15.5.3 Property overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Properties</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>542</td>
<td>Email</td>
<td>rw</td>
<td>Email of the server administrator</td>
</tr>
<tr>
<td>542</td>
<td>Name</td>
<td>rw</td>
<td>Name of the server administrator</td>
</tr>
<tr>
<td>541</td>
<td>Names</td>
<td>r</td>
<td>Indexed array with available variable names.</td>
</tr>
<tr>
<td>540</td>
<td>Values</td>
<td>r</td>
<td>Variables passed to the CGI script</td>
</tr>
<tr>
<td>542</td>
<td>VariableCount</td>
<td>r</td>
<td>Number of available variables.</td>
</tr>
<tr>
<td>541</td>
<td>Variables</td>
<td>r</td>
<td>Indexed array with variables as name=value pairs.</td>
</tr>
</tbody>
</table>

15.5.4 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 (541) property.

See also: Variables (541), Names (541), Values (540)

15.5.5 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 (538)

15.5.6 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 (541) property) and calls the DoPost (540) or DoGet (540) 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 (541), DoPost (540), DoGet (540)

15.5.7 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 (540) or DoGet (540) methods.

See also: DoPost (540), DoGet (540), PutLine (539)

15.5.8 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 (539) 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 (539)
15.5.9 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 (540)

15.5.10 TEZcgi.DoPost

Synopsis: Method to handle POST requests

Declaration: procedure DoPost; Virtual

Visibility: public

Description: DoPost is called by the Run (539) 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 (539), DoGet (540)

15.5.11 TEZcgi.DoGet

Synopsis: Method to handle GET requests

Declaration: procedure DoGet; Virtual

Visibility: public

Description: DoGet is called by the Run (539) 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 (539), DoPost (540)

15.5.12 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

540
CHAPTER 15. REFERENCE FOR UNIT ‘EZCGI’

GATEWAY_INTERFACE Used gateway interface
PATH_INFO Requested URL
PATH_TRANSLATED Transformed URL
QUERY_STRING Client query string
REMOTE_ADDR Address of remote client
REMOTE_HOST DNS name of remote client
REMOTE_IDENT Remote identity.
REMOTE_USER Remote user
REQUEST_METHOD Request methods (POST or GET)
SCRIPT_NAME Script name
SERVER_NAME Server host name
SERVER_PORT Server port
SERVER_PROTOCOL Server protocol
SERVER_SOFTWARE Web server software
HTTP_ACCEPT Accepted responses
HTTP_ACCEPT_CHARSET Accepted character sets
HTTP_ACCEPT_ENCODING Accepted encodings
HTTP_IF_MODIFIED_SINCE Proxy information
HTTP_REFERER Referring page
HTTP_USER_AGENT Client software name

Other than the standard list, any variables that were passed by the web-client request, are also available. Note that the variables are case insensitive.

See also: TEZCGI.Variables (541), TEZCGI.Names (541), TEZCGI.GetValue (540), TEZcgi.VariableCount (542)

15.5.13 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 (542). Any other value will result in an exception being raised.

See also: TEZcgi.Variables (541), TEZcgi.Values (540), TEZcgi.GetValue (540), TEZcgi.VariableCount (542)

15.5.14 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 (542). Any other value will result in an exception being raised.

See also: TEZcgi-names (541), TEZcgi-values (540), TEZcgi GetValue (540), TEZcgiVariableCount (542)

15.5.15 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 (541) and Variables (541) properties.

See also: Names (541), Variables (541), TEZcgi-values (540), TEZcgi.GetValue (540)

15.5.16 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 (539) method.

See also: TEZcgi.Run (539), TEZcgi.Email (542)

15.5.17 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 (539) method.

See also: TEZcgi.Run (539), TEZcgi.Name (542)
Chapter 16

Reference for unit ’fpjson’

16.1 Used units

Table 16.1: Used units by unit ’fpjson’

<table>
<thead>
<tr>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Classes</td>
<td>??</td>
</tr>
<tr>
<td>Contnrs</td>
<td>201</td>
</tr>
<tr>
<td>System</td>
<td>??</td>
</tr>
<tr>
<td>sysutils</td>
<td>??</td>
</tr>
<tr>
<td>Variants</td>
<td>??</td>
</tr>
</tbody>
</table>

16.2 Overview

The JSON unit implements JSON support for Free Pascal. It contains the data structures (TJSONData (567) and descendent objects) to treat JSON data and output JSON as a string TJSONData.AsJSON (576). The generated JSON can be formatted in several ways TJSONData.FormatJSON (572).

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 (579), TJSONFloatNumber (577) or TJSONInt64Number (578), depending on the type of the number.

**Strings** in TJSONString (594).

**Boolean** in TJSONBoolean (566).

**null** is supported using TJSONNull (580)

**Array** is supported using TJSONArray (555)

**Object** is supported using TJSONObject (582)

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:
CHAPTER 16. REFERENCE FOR UNIT ‘FPJSON’

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 (553) and then the GetJSON (551) 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.

declare will print "Maria".

The FPJSON code does not use hardcoded class names when creating the JSON: it uses the various CreateJSON (550) 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 (551) and SetJSONInstanceType (552) functions can be used to get or set the classes that must be used. the default parser used by GetJSON (551) will also use these functions.
16.3 Constants, types and variables

16.3.1 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 (572) behave as TJSONData.AsJSON (576) with TJSONData.CompressedJSON equal to True and TJSONData.UnquotedMemberNames equal to True.

AsCompressedJSON = [foSingleLineArray, foSingleLineObject, foskipWhiteSpace]

AsCompressedJSON can be used to let TJSONData.FormatJSON (572) behave as TJSONData.AsJSON (576) with TJSONData.CompressedJSON (572) equal to True

AsJSONFormat = [foSingleLineArray, foSingleLineObject]

AsJSONFormat contains the options that make TJSONData.FormatJSON (572) behave like TJSONData.AsJSON (576)

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.
16.3.2 Types

PJSONCharType = ^TJSONCharType

PJSONCharType is a pointer to a TJSONCharType (546) character. It is used while parsing JSON.

TFormatOption = (foSingleLineArray, foSingleLineObject,
foDoNotQuoteMembers, foUseTabchar, foSkipWhiteSpace,
foSkipWhiteSpaceOnlyLeading)

Table 16.2: Enumeration values for type TFormatOption

<table>
<thead>
<tr>
<th>Value</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>foDoNotQuoteMembers</td>
<td>Do not use quote characters around object member names.</td>
</tr>
<tr>
<td>foSingleLineArray</td>
<td>Keep all array elements on a single line.</td>
</tr>
<tr>
<td>foSingleLineObject</td>
<td>Keep all object elements on a single line.</td>
</tr>
<tr>
<td>foSkipWhiteSpace</td>
<td>Skip whitespace</td>
</tr>
<tr>
<td>foSkipWhiteSpaceOnlyLeading</td>
<td>Only skip leading whitespace when formatting JSON</td>
</tr>
<tr>
<td>foUseTabchar</td>
<td>Use the tabulator character for indents</td>
</tr>
</tbody>
</table>

TFormatOption enumerates the various formatting options that can be used in the TJSONData.FormatJSON (572) function.

TFormatOptions = Set of TFormatOption

TFormatOptions is the set definition used to specify options in TJSONData.FormatJSON (572).

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 (555) class. It is used in CreateJSONArray (550).

TJSONArrayIterator = procedure(Item: TJSONData; Data: TObject;
var Continue: Boolean) of object

TJSONArrayIterator is the procedural callback used by TJSONArray.Iterate (557) 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 (566). It is used in the factory methods.

TJSONCharType = AnsiChar

TJSONCharType is the type of a single character in a TJSONStringType (549) string. It is used by the parser.
TJSONDataClass = Class of TJSONData

TJSONDataClass is used in the CreateJSON (550), SetJSONInstanceType (552) and GetJSONInstanceType (551) 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 (577). It is used in the factory methods.

TJSONInstanceType = (jitUnknown, jitNumberInteger, jitNumberInt64, jitNumberQWord, jitNumberFloat, jitString, jitBoolean, jitNull, jitArray, jitObject)

Table 16.3: Enumeration values for type TJSONInstanceType

<table>
<thead>
<tr>
<th>Value</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>jitArray</td>
<td>Array value</td>
</tr>
<tr>
<td>jitBoolean</td>
<td>Boolean value</td>
</tr>
<tr>
<td>jitNull</td>
<td>Null value</td>
</tr>
<tr>
<td>jitNumberFloat</td>
<td>Floating point real number</td>
</tr>
<tr>
<td>jitNumberInt64</td>
<td>64-bit signed integer number</td>
</tr>
<tr>
<td>jitNumberInteger</td>
<td>32-bit signed integer number</td>
</tr>
<tr>
<td>jitNumberQWord</td>
<td>Qword integer number type</td>
</tr>
<tr>
<td>jitObject</td>
<td>Object value</td>
</tr>
<tr>
<td>jitString</td>
<td>String value</td>
</tr>
<tr>
<td>jitUnknown</td>
<td>Unknown</td>
</tr>
</tbody>
</table>

TJSONInstanceType is used by the parser to determine what kind of TJSONData (567) descendant to create for a particular data item. It is a more fine-grained division than TJSONType (549).

TJSONInt64NumberClass = Class of TJSONInt64Number

TJSONInt64NumberClass is the class type of TJSONInt64Number (578). It is used in the factory methods.

TJSONIntegerNumberClass = Class of TJSONIntegerNumber

TJSONIntegerNumberClass is the class type of TJSONIntegerNumber (579). 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 (548)

TJSONNullClass = Class of TJSONNull

TJSONNullClass is the class type of TJSONNull (580). It is used in the factory methods.

TJSONNumberType = (ntFloat, ntInteger, ntInt64, ntQWord)

Table 16.4: Enumeration values for type TJSONNumberType

<table>
<thead>
<tr>
<th>Value</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>ntFloat</td>
<td>Floating point value</td>
</tr>
<tr>
<td>ntInt64</td>
<td>64-bit integer value</td>
</tr>
<tr>
<td>ntInteger</td>
<td>32-bit Integer value</td>
</tr>
<tr>
<td>ntQWord</td>
<td>64-bit unsigned integer value</td>
</tr>
</tbody>
</table>

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 (582) value.

TJSONObjectClass = Class of TJSONObject

TJSONObject is the class type for the TJSONObject (582) class. It is used in CreateJSONObject (550).

TJSONObjectIterator = procedure(const AName: TJSONStringType; Item: TJSONData; Data: TObject; var Continue: Boolean) of object

TJSONObjectIterator is the procedural callback used by TJSONObject.Iterate (585) 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 (551) 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 (552) method, the actual types can be retrieved with GetJSONInstanceType (551)
CHAPTER 16. REFERENCE FOR UNIT 'FPJSON'

TJSONQWordNumberClass = Class of TJSONQWordNumber

TJSONQWordNumberClass is the class type of TJSONQWordNumber (593). It is used in the factory methods.

TJSONStringClass = Class of TJSONString

TJSONStringClass is the class type of TJSONString (594). 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 convert a JSON string to a TJSONData (567). It is used in the SetJSONStringParserHandler (553) and GetJSONStringParserHandler (552) 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 16.5: Enumeration values for type TJSONtype

<table>
<thead>
<tr>
<th>Value</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>jtArray</td>
<td>Array data (integer index, elements can be any type)</td>
</tr>
<tr>
<td>jtBoolean</td>
<td>Boolean data</td>
</tr>
<tr>
<td>jtNull</td>
<td>Null data</td>
</tr>
<tr>
<td>jtNumber</td>
<td>Numerical type. This can be integer (32/64 bit) or float.</td>
</tr>
<tr>
<td>jtObject</td>
<td>Object data (named index, elements can be any type)</td>
</tr>
<tr>
<td>jtString</td>
<td>String data type.</td>
</tr>
<tr>
<td>jtUnknown</td>
<td>Unknown JSON data type</td>
</tr>
</tbody>
</table>

TJSONtype determines the type of JSON data a particular object contains. The class function TJSONData.JSONType (568) returns this type, and indicates what kind of data that particular descendant 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 (574) property.

TJSONVariant = variant

TJSONVariant resolves to Variant on native platforms, JSValue in javascript runtimes.

549
16.4 Procedures and functions

16.4.1 CreateJSON

Synopsis: Create a JSON data item

Declaration:

```plaintext
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(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 (551) to do so. The classes to be used can be set using the SetJSONInstanceType (552).

The JSON parser uses these functions to create instances of TJSONData (567).

Errors: None.

See also: GetJSONInstanceType (551), SetJSONInstanceType (552), GetJSON (551), CreateJSONArray (550), CreateJSONObject (550)

16.4.2 CreateJSONArray

Synopsis: Create a JSON array

Declaration:

```plaintext
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 (551), SetJSONInstanceType (552), GetJSON (551), CreateJSON (550), TJSONArray (555)

16.4.3 CreateJSONObject

Synopsis: Create a JSON object

Declaration:

```plaintext
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 (583) 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 (551), SetJSONInstanceType (552), GetJSON (551), CreateJSON (550), TJSONObject (582)
16.4.4 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 (553) 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 (551), SetJSONParserHandler (553), TJSONData (567)

16.4.5 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 (567) 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 (550) 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 (550), TJSONData (567), GetJSON (551)

16.4.6 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 (553), GetJSONStringParserHandler (552), TJSONParserHandler (548), GetJSON (551)
16.4.7 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 (553) call.
See also: SetJSONStringParserHandler (553), SetJSONParserHandler (553), GetJSONParserHandler (551)

16.4.8 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 (553), JSONTypeName (552)

16.4.9 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 (553), JSONStringToString (552)

16.4.10 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 (567) 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 instances: 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 (550) 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 (551), CreateJSON (550)
16.4.11 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 (553) call.

See also: SetJSONStringParserHandler (553), GetJSONParserHandler (551), TJSONParserHandler (548), GetJSON (551)

16.4.12 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 (553). It sets a callback that will be used by the GetJSON (551) 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 (553) 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 (552), SetJSONParserHandler (553), GetJSONParserHandler (551)

16.4.13 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 converting string values to JSON. If set to False, will also be escaped, although this is strictly speaking not necessary.

See also: JSONStringToString (552), JSONTypeName (552)

16.5 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).

16.6 EJSON

16.6.1 Description

EJSON is the exception raised by the JSON implementation to report JSON error.

16.7 TBaseJSONEnumerator

16.7.1 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 (554)

16.7.2 Method overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>554</td>
<td>GetCurrent</td>
<td>Return the current value of the enumerator</td>
</tr>
<tr>
<td>555</td>
<td>MoveNext</td>
<td>Move to next value in array/object</td>
</tr>
</tbody>
</table>

16.7.3 Property overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Properties</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>555</td>
<td>Current</td>
<td>r</td>
<td>Return the current value of the enumerator</td>
</tr>
</tbody>
</table>

16.7.4 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 (554) value.

See also: TJSONEnum (554)

16.7.5 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 (554), TJSONData (567)

16.7.6 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 (554). It is only valid after MoveNext (555) returned True.

See also: TJSONEnum (554), TJSONData (567), MoveNext (555)

16.8 TJSONArray

16.8.1 Description

TJSONArrayClass is the class type of TJSONArray (555). It is used in the factory methods.

See also: TJSONArray (555), SetJSONInstanceType (552), GetJSONInstanceType (551)
16.8.2 Method overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>558</td>
<td>Add</td>
<td>Add a JSON value to the array</td>
</tr>
<tr>
<td>558</td>
<td>Clear</td>
<td>Clear the array</td>
</tr>
<tr>
<td>557</td>
<td>Clone</td>
<td>Clone the JSON array</td>
</tr>
<tr>
<td>556</td>
<td>Create</td>
<td>Create a new instance of JSON array data.</td>
</tr>
<tr>
<td>559</td>
<td>Delete</td>
<td>Delete an element from the list by index</td>
</tr>
<tr>
<td>557</td>
<td>Destroy</td>
<td>Free the JSON array</td>
</tr>
<tr>
<td>559</td>
<td>Exchange</td>
<td>Exchange 2 elements in the list</td>
</tr>
<tr>
<td>559</td>
<td>Extract</td>
<td>Extract an element from the array</td>
</tr>
<tr>
<td>558</td>
<td>GetEnumerator</td>
<td>Get an array enumerator</td>
</tr>
<tr>
<td>558</td>
<td>IndexOf</td>
<td>Return index of JSONData instance in array</td>
</tr>
<tr>
<td>560</td>
<td>Insert</td>
<td>Insert an element in the array</td>
</tr>
<tr>
<td>557</td>
<td>Iterate</td>
<td>Iterate over all elements in the array</td>
</tr>
<tr>
<td>557</td>
<td>JSONType</td>
<td>native JSON data type</td>
</tr>
<tr>
<td>560</td>
<td>Move</td>
<td>Move a value from one location to another</td>
</tr>
<tr>
<td>560</td>
<td>Remove</td>
<td>Remove an element from the list</td>
</tr>
<tr>
<td>561</td>
<td>Sort</td>
<td>Sort the items in the array</td>
</tr>
</tbody>
</table>

16.8.3 Property overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Properties</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>565</td>
<td>Arrays</td>
<td>rw</td>
<td>Get or set elements as JSON array values</td>
</tr>
<tr>
<td>565</td>
<td>Booleans</td>
<td>rw</td>
<td>Get or set elements as boolean values</td>
</tr>
<tr>
<td>564</td>
<td>Floats</td>
<td>rw</td>
<td>Get or set elements as floating-point numerical values</td>
</tr>
<tr>
<td>562</td>
<td>Int64s</td>
<td>rw</td>
<td>Get or set elements as Int64 values</td>
</tr>
<tr>
<td>562</td>
<td>Integers</td>
<td>rw</td>
<td>Get or set elements as integer values</td>
</tr>
<tr>
<td>561</td>
<td>Items</td>
<td></td>
<td>Indexed access to the values in the array</td>
</tr>
<tr>
<td>563</td>
<td>LargeInts</td>
<td>rw</td>
<td>Get or set elements as LargeInt values</td>
</tr>
<tr>
<td>561</td>
<td>Nulls</td>
<td>r</td>
<td>Check which elements are null</td>
</tr>
<tr>
<td>565</td>
<td>Objects</td>
<td>rw</td>
<td>Get or set elements as JSON object values</td>
</tr>
<tr>
<td>563</td>
<td>QWords</td>
<td>rw</td>
<td>Get or set elements as QWord values</td>
</tr>
<tr>
<td>564</td>
<td>Strings</td>
<td>rw</td>
<td>Get or set elements as string values</td>
</tr>
<tr>
<td>561</td>
<td>Types</td>
<td>r</td>
<td>JSON types of elements in the array</td>
</tr>
<tr>
<td>563</td>
<td>UnicodeStrings</td>
<td>rw</td>
<td>Get or set elements as Unicode string values</td>
</tr>
</tbody>
</table>

16.8.4 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 (567) 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 (551)
16.8.5 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 (555) instance.

See also: TJSONArray.Clear (558), TJSONArray.Create (556)

16.8.6 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 (568)

16.8.7 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 (572)

16.8.8 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 (546)
16.8.9 **TJSONArray.IndexOf**

**Synopsis:** Return index of JSONData instance in array

**Declaration:**

```delphi
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 (558)

16.8.10 **TJSONArray.GetEnumerator**

**Synopsis:** Get an array enumerator

**Declaration:**

```delphi
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` (554).

See also: `TJSONEnum` (554), `TJSONData.GetEnumerator` (569)

16.8.11 **TJSONArray.Clear**

**Synopsis:** Clear the array

**Declaration:**

```delphi
procedure Clear; Override
```

**Visibility:** public

**Description:**

Clear clears the array and frees all elements in it. After the call to clear, Count (573) returns 0.

See also: Delete (559), Extract (559)

16.8.12 **TJSONArray.Add**

**Synopsis:** Add a JSON value to the array

**Declaration:**

```delphi
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 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 (559), Extract (559)

16.8.13 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 element is freed.

Errors: If an invalid index is passed, an exception is raised.

See also: Clear (558), Add (558), Extract (559), Exchange (559)

16.8.14 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.

16.8.15 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 (559) does. In difference with Delete, it does not free the object instance. Instead, it returns the extracted element.

See also: Delete (559), Clear (558), Insert (560), Add (558)
16.8.16  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 (558), it converts plain pascal values to JSON values. Note that when inserting a TJSONData (567) 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 (558), Delete (559), Extract (559), Clear (558)

16.8.17  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 (559), Extract (559), Insert (560)

16.8.18  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 (559), Extract (559)
16.8.19 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 (??)

16.8.20 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 (573). TJSONArray simply declares it as the default property.
See also: TJSONData.Items (573)

16.8.21 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 (568) result of the elements in the array. Accessing it is equivalent to accessing

Items[Index].JSONType

See also: TJSONData.JSONType (568), TJSONData.Items (573)

16.8.22 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 (576) property when reading. It is then equivalent to accessing
16.8.23 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 (575) 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 (573), TJSONData.IsNull (576), TJSONArray.Types (561), TJSONArray.Integers (562), TJSONArray.QWords (563), TJSONArray.Floats (564), TJSONArray.Strings (564), TJSONArray.Booleans (565)

16.8.24 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 (574) 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 (573), TJSONData.IsNull (576), TJSONArray.Types (561), TJSONArray.Integers (562), TJSONArray.Floats (564), TJSONArray.Strings (564), TJSONArray.Booleans (565), TJSONArray.QWords (563)
16.8.25 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 (575) property when reading. Reading it is the equivalent to accessing

\[ \text{Items[Index].AsLargeInt} \]

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 (575), TJSONData.Items (573), TJSONData.IsNull (576), TJSONArray.Types (561), TJSONArray.Integers (562), TJSONArray.Floats (564), TJSONArray.Strings (564), TJSONArray.Booleans (565), TJSONArray.QWords (563)

16.8.26 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 (574) property when reading. Reading it is the equivalent to accessing

\[ \text{Items[Index].AsQWord} \]

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 (543), AsQWord (574), IsNull (576), Types (561), Integers (562), Floats (564), Strings (564), Booleans (565)

16.8.27 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 (574) property when reading. Reading it is the equivalent to accessing
CHAPTER 16. REFERENCE FOR UNIT ‘FPJSON’

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 (573), TJSONData.IsNull (576), TJSONArray.Types (561), TJSONArray.Integers (562), TJSONArray.QWords (563), TJSONArray.Floats (564), TJSONArray.Int64s (562), TJSONArray.Booleans (565), TJSONArray.Strings (564)

16.8.28 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 (573) 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 (573), TJSONData.IsNull (576), TJSONArray.Types (561), TJSONArray.Integers (562), TJSONArray.QWords (563), TJSONArray.Floats (564), TJSONArray.Int64s (562), TJSONArray.Booleans (565)

16.8.29 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 (575) 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 (573), TJSONData.IsNull (576), TJSONArray.Types (561), TJSONArray.Integers (562), TJSONArray.QWords (563), TJSONArray.Floats (564), TJSONArray.Int64s (562), TJSONArray.Booleans (565)
16.8.30 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 (576) 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 (573), TJSONData.IsNull (576), TJSONArray.Types (561), TJSONArray.Integers (562), TJSONArray.Strings (564), TJSONArray.Int64s (562), TJSONArray.QWords (563), TJSONArray.Floats (564)

16.8.31 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 (573), TJSONData.IsNull (576), TJSONArray.Types (561), TJSONArray.Integers (562), TJSONArray.Strings (564), TJSONArray.Int64s (562), TJSONArray.QWords (563), TJSONArray.Floats (564), TJSONArray.Objects (565)

16.8.32 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 (573), TJSONData.IsNull (576), TJSONArray.Types (561), TJSONArray.Integers (562), TJSONArray.Strings (564), TJSONArray.Int64s (562), TJSONArray.QWords (563), TJSONArray.Floats (564), TJSONArray.Arrays (565)

16.9 TJSONBoolean

16.9.1 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 (577), TJSONIntegerNumber (579), TJSONInt64Number (578), TJSONBoolean (566), TJSONNull (580), TJSONArray (555), TJSONObject (582)

16.9.2 Method overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>567</td>
<td>Clear</td>
<td>Clear data</td>
</tr>
<tr>
<td>567</td>
<td>Clone</td>
<td>Clone boolean value</td>
</tr>
<tr>
<td>566</td>
<td>Create</td>
<td>Create a new instance of boolean JSON data</td>
</tr>
<tr>
<td>566</td>
<td>JSONType</td>
<td>native JSON data type</td>
</tr>
</tbody>
</table>

16.9.3 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 (580), TJSONFloatNumber.Create (577), TJSONInt64Number.Create (578), TJSONString.Create (594), TJSONArray.Create (556), TJSONObject.Create (583)

16.9.4 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 (568)
16.9.5 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 (569)

16.9.6 TJSONBoolean.Clone

Synopsis: Clone boolean value

Declaration: function Clone : TJSONData; Override

Visibility: public

Description: Clone overrides TJSONData.Clone (572) and creates an instance of the same class with the same boolean value.

See also: TJSONData.Clone (572)

16.10 TJSONData

16.10.1 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 (579), TJSONFloatNumber (577) or TJSONInt64Number (578), depending on the type of the number.

**Strings** can be represented with TJSONString (594).

**Boolean** can be represented with TJSONBoolean (566).

**null** is supported using TJSONNull (580)

**Array** data can be represented using TJSONArray (555)

**Object** data can be supported using TJSONObject (582)

See also: TJSONIntegerNumber (579), TJSONString (594), TJSONBoolean (566), TJSONNull (580), TJSONArray (555), TJSONObject (582)
16.10.2 Method overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>569</td>
<td>Clear</td>
<td>Clear the raw value of this data object</td>
</tr>
<tr>
<td>572</td>
<td>Clone</td>
<td>Duplicate the value of the JSON data</td>
</tr>
<tr>
<td>568</td>
<td>Create</td>
<td>Create a new instance of TJSONData.</td>
</tr>
<tr>
<td>569</td>
<td>DumpJSON</td>
<td>Fast, memory efficient dump of JSON in stream</td>
</tr>
<tr>
<td>569</td>
<td>FindPath</td>
<td>Find data by name</td>
</tr>
<tr>
<td>572</td>
<td>FormatJSON</td>
<td>Return a formatted JSON representation of the data.</td>
</tr>
<tr>
<td>569</td>
<td>GetEnumerator</td>
<td>Return an enumerator for the data</td>
</tr>
<tr>
<td>571</td>
<td>GetPath</td>
<td>Get data by name</td>
</tr>
<tr>
<td>568</td>
<td>JSONType</td>
<td>The native JSON data type represented by this object</td>
</tr>
</tbody>
</table>

16.10.3 Property overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Properties</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>576</td>
<td>AsBoolean</td>
<td>rw</td>
<td>Access the raw JSON value as a boolean</td>
</tr>
<tr>
<td>575</td>
<td>AsFloat</td>
<td>rw</td>
<td>Access the raw JSON value as a float</td>
</tr>
<tr>
<td>574</td>
<td>AsInt64</td>
<td>rw</td>
<td>Access the raw JSON value as an 64-bit integer</td>
</tr>
<tr>
<td>575</td>
<td>AsInteger</td>
<td>rw</td>
<td>Access the raw JSON value as a 32-bit integer</td>
</tr>
<tr>
<td>576</td>
<td>AsJSON</td>
<td>r</td>
<td>Return a JSON representation of the value</td>
</tr>
<tr>
<td>575</td>
<td>AsLargeInt</td>
<td>rw</td>
<td>Access to data as largeint</td>
</tr>
<tr>
<td>574</td>
<td>AsQWord</td>
<td>rw</td>
<td>Access the raw JSON value as an 64-bit unsigned integer</td>
</tr>
<tr>
<td>573</td>
<td>AsString</td>
<td>rw</td>
<td>Access the raw JSON value as a string</td>
</tr>
<tr>
<td>574</td>
<td>AsUnicodeString</td>
<td>rw</td>
<td>Return the value as a Unicode string</td>
</tr>
<tr>
<td>572</td>
<td>CompressedJSON</td>
<td>rw</td>
<td>Compress JSON - skip whitespace</td>
</tr>
<tr>
<td>573</td>
<td>Count</td>
<td>r</td>
<td>Number of sub-items for this data element</td>
</tr>
<tr>
<td>576</td>
<td>IsNull</td>
<td>r</td>
<td>Is the data a null value ?</td>
</tr>
<tr>
<td>573</td>
<td>Items</td>
<td>rw</td>
<td>Indexed access to sub-items</td>
</tr>
<tr>
<td>573</td>
<td>Value</td>
<td>rw</td>
<td>The value of this data object as a variant</td>
</tr>
</tbody>
</table>

16.10.4 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 (549)

16.10.5 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.
16.10.6 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 (568)

16.10.7 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 (576) and writing it to stream.

Errors: None.

See also: TJSONData.FormatJSON (572), TJSONData.AsJSON (576)

16.10.8 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 (567) type, but a TJSONEnum (554) structure, which contains the value, and for structured types, the key (numerical or string).

See also: TJSONEnum (554), TJSONArray (555), TJSONObject (582)

16.10.9 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 (§82), a member can be specified using its name, and the object value itself can be retrieved with the empty path.
- For JSON Arrays (§82), 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:

```pascal
Var D,E : TJSONData;
begin
D:=TJSONIntegerNumber.Create(123);
E:=D.FindPath('');
end.
```

The following code will not return anything:

```pascal
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:

```pascal
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:

```pascal
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:

```pascal
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 (555), TJSONObject (582), GetPath (571)

### 16.10.10 TJSONData.GetPath

Synopsis: Get data by name

Declaration: `function GetPath(const APath: TJSONStringType) : TJSONData`  

Visibility: public

Description: GetPath is identical to FindPath (569) but raises an exception if no element was found. The exception message contains the piece of path that was not found.

Errors: An EJSON (554) exception is raised if the path does not exist.

See also: FindPath (569)
16.10.11 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 (569), EOutOfMemory (??)

16.10.12 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 property, but for complex values (TJSONArray (555) and TJSONObject (582)) 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 (576), TFormatOptions (546)

16.10.13 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 (576) 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 (572), TJSONData.AsJSON (576), TJSONData.UnquotedMemberNames (567), TJSONData.AsCompressedJSON (567), TJSONObject.UnquotedMemberNames (588)

572
16.10.14 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 (573)

16.10.15 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 (555) and TJSONObject (582)) override this method to return the Index-th element in the list.

See also: Count (573), TJSONArray (555), TJSONObject (582)

16.10.16 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 (549)), 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 (549)

16.10.17 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 (594) this will return the native value.

For complex values, reading or writing this property will result in an EConvertError exception.

See also: AsInteger (575), Value (573), AsInt64 (574), AsFloat (575), AsBoolean (576), IsNull (576), AsJSON (576)

16.10.18 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 (573)

16.10.19 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 (578) this will return the native value.

For complex values, reading or writing this property will always result in an EConvertError exception.

See also: AsFloat (575), Value (573), AsInteger (575), AsString (573), AsBoolean (576), IsNull (576), AsJSON (576)

16.10.20 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 (593) this will return the native value.
For complex values, reading or writing this property will always result in an EConvertError exception.

See also: AsFloat (575), Value (573), AsInteger (575), AsInt64 (574), AsString (573), AsBoolean (576), IsNull (576), AsJSON (576)

16.10.21 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: NativeInt on the pas2JS platform, Int64 on all other platforms.

16.10.22 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 (577) this will return the native value.
For complex values, reading or writing this property will always result in an EConvertError exception.

See also: AsInteger (575), Value (573), AsInt64 (574), AsString (573), AsBoolean (576), IsNull (576), AsJSON (576)

16.10.23 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 (579) this will return the native value.

For complex values, reading or writing this property will always result in an EConvertError exception.

See also: AsFloat (575), Value (573), AsInt64 (574), AsString (573), AsBoolean (576), IsNull (576), AsJSON (576)

16.10.24 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 (566) this will return the native value.

For complex values, reading or writing this property will always result in an EConvertError exception.

See also: AsFloat (575), Value (573), AsInt64 (574), AsString (573), AsInteger (575), IsNull (576), AsJSON (576)

16.10.25 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 (580) instance. In all other cases, it is False. This value cannot be set.

See also: TJSONType (549), JSONType (568), TJSONNull (580), AsFloat (575), Value (573), AsInt64 (574), AsString (573), AsInteger (575), AsBoolean (576), AsJSON (576)

16.10.26 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 (575)`, `Value (573)`, `AsInt64 (574)`, `AsString (573)`, `AsInteger (575)`, `AsBoolean (576)`, `AsJSON (576)`

## 16.11 TJSONFloatNumber

### 16.11.1 Description

`TJSONFloatNumber` must be used whenever floating point data must be represented. It can handle `TJSONFloat (547)` data (normally a double). For integer data, `TJSONIntegerNumber (579)` or `TJSONInt64Number (578)` are better suited.

See also: `TJSONNumber (581)`, `TJSONFloat (547)`, `TJSONIntegerNumber (579)`, `TJSONInt64Number (578)`

### 16.11.2 Method overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>578</td>
<td>Clear</td>
<td>Clear value</td>
</tr>
<tr>
<td>578</td>
<td>Clone</td>
<td>Clone floating point value</td>
</tr>
<tr>
<td>577</td>
<td>Create</td>
<td>Create a new floating-point value</td>
</tr>
<tr>
<td>577</td>
<td>NumberType</td>
<td>Kind of numerical data managed by this class.</td>
</tr>
</tbody>
</table>

### 16.11.3 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 (580)`, `TJSONInt64Number.Create (578)`

### 16.11.4 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 (548)`, `TJSONData.JSONtype (568)`
16.11.5 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 (569)

16.11.6 TJSONFloatNumber.Clone

Synopsis: Clone floating point value

Declaration: function Clone : TJSONData; Override

Visibility: public

Description: Clone overrides TJSONData.Clone (572) and creates an instance of the same class with the same floating-point value.

See also: TJSONData.Clone (572)

16.12 TJSONInt64Number

16.12.1 Description

TJSONInt64Number must be used whenever 64-bit integer data must be represented. For 32-bit integer data, TJSONIntegerNumber (579) must be used.

See also: TJSONFloatNumber (577), TJSONIntegerNumber (579)

16.12.2 Method overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>579</td>
<td>Clear</td>
<td>Clear value</td>
</tr>
<tr>
<td>579</td>
<td>Clone</td>
<td>Clone 64-bit integer value</td>
</tr>
<tr>
<td>578</td>
<td>Create</td>
<td>Create a new instance of 64-bit integer JSON data</td>
</tr>
<tr>
<td>579</td>
<td>NumberType</td>
<td>Kind of numerical data managed by this class.</td>
</tr>
</tbody>
</table>

16.12.3 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 (580), TJSONFloatNumber.Create (577)
16.12.4  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 (548), TJSONData.JSONtype (568)

16.12.5  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 (569)

16.12.6  TJSONInt64Number.Clone
Synopsis: Clone 64-bit integer value
Declaration: function Clone : TJSONData; Override
Visibility: public
Description: Clone overrides TJSONData.Clone (572) and creates an instance of the same class with the same
64-bit integer value.
See also: TJSONData.Clone (572)

16.13  TJSONIntegerNumber

16.13.1  Description
TJSONIntegerNumber must be used whenever 32-bit integer data must be represented. For 64-bit integer data, TJSONInt64Number (578) must be used.

See also: TJSONFloatNumber (577), TJSONInt64Number (578)

16.13.2  Method overview
<table>
<thead>
<tr>
<th>Page</th>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>580</td>
<td>Clear</td>
<td>Clear value</td>
</tr>
<tr>
<td>580</td>
<td>Clone</td>
<td>Clone 32-bit integer value</td>
</tr>
<tr>
<td>580</td>
<td>Create</td>
<td>Create a new instance of 32-bit integer JSON data</td>
</tr>
<tr>
<td>580</td>
<td>NumberType</td>
<td>Kind of numerical data managed by this class.</td>
</tr>
</tbody>
</table>
16.13.3 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 (577), TJSONInt64Number.Create (578)

16.13.4 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 (548), TJSONData.JSONtype (568)

16.13.5 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 (569)

16.13.6 TJSONIntegerNumber.Clone

Synopsis: Clone 32-bit integer value

Declaration: function Clone: TJSONData; Override

Visibility: public

Description: Clone overrides TJSONData.Clone (572) and creates an instance of the same class with the same 32-bit integer value.

See also: TJSONData.Clone (572)

16.14 TJSONNull

16.14.1 Description

TJSONNull must be used whenever a null value must be represented.

See also: TJSONFloatNumber (577), TJSONIntegerNumber (579), TJSONInt64Number (578), TJSONBoolean (566), TJSONString (594), TJSONArray (555), TJSONObject (582)
16.14.2 Method overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>581</td>
<td>Clear</td>
<td>Clear data</td>
</tr>
<tr>
<td>581</td>
<td>Clone</td>
<td>Clone boolean value</td>
</tr>
<tr>
<td>581</td>
<td>JSONType</td>
<td>native JSON data type</td>
</tr>
</tbody>
</table>

16.14.3 TJSONObject.JSONType

Synopsis: native JSON data type

Declaration: class function JSONType : TJSONObject; Override

Visibility: public

Description: JSONType is overridden by TJSONNull to return jtNull.

See also: TJSONData.JSONType (568)

16.14.4 TJSONNull.Clear

Synopsis: Clear data

Declaration: procedure Clear; Override

Visibility: public

Description: Clear does nothing.

See also: TJSONData.Clear (569)

16.14.5 TJSONNull.Clone

Synopsis: Clone boolean value

Declaration: function Clone : TJSONData; Override

Visibility: public

Description: Clone overrides TJSONData.Clone (572) and creates an instance of the same class.

See also: TJSONData.Clone (572)

16.15 TJSONNumber

16.15.1 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 (579), TJSONInt64Number (578) or TJSONFloatNumber (577) should be instantiated.

See also: TJSONIntegerNumber (579), TJSONInt64Number (578), TJSONFloatNumber (577)
### 16.15.2 Method overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>582</td>
<td>JSONType</td>
<td>native JSON data type</td>
</tr>
<tr>
<td>582</td>
<td>NumberType</td>
<td>Kind of numerical data managed by this class.</td>
</tr>
</tbody>
</table>

### 16.15.3 TJSONNumber.JSONType

**Synopsis:** native JSON data type

**Declaration:**
```pascal
class function JSONType : TJSONtype; Override
```

**Visibility:** public

**Description:**

JSONType is overridden by TJSONNumber to return jtNumber.

See also: TJSONData.JSONType (568)

### 16.15.4 TJSONNumber.NumberType

**Synopsis:** Kind of numerical data managed by this class.

**Declaration:**
```pascal
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 (579), TJSONInt64Number (578), TJSONFloatNumber (577), JSONType (543)

### 16.16 TJSONObject

### 16.16.1 Description

TJSONObjectClass is the class type of TJSONObject (582). It is used in the factory methods.

See also: TJSONObject (582), SetJSONInstanceType (552), GetJSONInstanceType (551)
16.16.2 Method overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>587</td>
<td>Add</td>
<td>Add a name, value to the object</td>
</tr>
<tr>
<td>587</td>
<td>Clear</td>
<td>Clear the object</td>
</tr>
<tr>
<td>584</td>
<td>Clone</td>
<td>Clone the JSON object</td>
</tr>
<tr>
<td>583</td>
<td>Create</td>
<td>Create a new instance of JSON object data.</td>
</tr>
<tr>
<td>587</td>
<td>Delete</td>
<td>Delete an element from the list by index</td>
</tr>
<tr>
<td>584</td>
<td>Destroy</td>
<td>Free the JSON object</td>
</tr>
<tr>
<td>588</td>
<td>Extract</td>
<td>Extract an element from the object</td>
</tr>
<tr>
<td>586</td>
<td>Find</td>
<td>Find an element by name.</td>
</tr>
<tr>
<td>586</td>
<td>Get</td>
<td>Retrieve a value by name</td>
</tr>
<tr>
<td>585</td>
<td>GetEnumerator</td>
<td>Get an object enumerator</td>
</tr>
<tr>
<td>585</td>
<td>IndexOf</td>
<td>Return index of JSONData instance in object</td>
</tr>
<tr>
<td>585</td>
<td>IndexOfName</td>
<td>Return index of name in item list</td>
</tr>
<tr>
<td>585</td>
<td>Iterate</td>
<td>Iterate over all elements in the object</td>
</tr>
<tr>
<td>584</td>
<td>JSONType</td>
<td>native JSON data type</td>
</tr>
<tr>
<td>588</td>
<td>Remove</td>
<td>Remove item by instance</td>
</tr>
</tbody>
</table>

16.16.3 Property overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Properties</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>592</td>
<td>Arrays</td>
<td>rw</td>
<td>Named access to JSON array values</td>
</tr>
<tr>
<td>592</td>
<td>Booleans</td>
<td>rw</td>
<td>Named access to boolean values</td>
</tr>
<tr>
<td>589</td>
<td>Elements</td>
<td>rw</td>
<td>Name-based access to JSON values in the object.</td>
</tr>
<tr>
<td>590</td>
<td>Floats</td>
<td>rw</td>
<td>Named access to float values</td>
</tr>
<tr>
<td>590</td>
<td>Int64s</td>
<td>rw</td>
<td>Named access to int64 values</td>
</tr>
<tr>
<td>590</td>
<td>Integers</td>
<td>rw</td>
<td>Named access to integer values</td>
</tr>
<tr>
<td>591</td>
<td>LargeInts</td>
<td>rw</td>
<td>Get or set elements as LargeInt values</td>
</tr>
<tr>
<td>589</td>
<td>Names</td>
<td>r</td>
<td>Indexed access to the names of elements.</td>
</tr>
<tr>
<td>590</td>
<td>Nulls</td>
<td>rw</td>
<td>Named access to null values</td>
</tr>
<tr>
<td>593</td>
<td>Objects</td>
<td>rw</td>
<td>Named access to JSON object values</td>
</tr>
<tr>
<td>591</td>
<td>QWords</td>
<td>rw</td>
<td>Named access to QWord values</td>
</tr>
<tr>
<td>592</td>
<td>Strings</td>
<td>rw</td>
<td>Named access to string values</td>
</tr>
<tr>
<td>589</td>
<td>Types</td>
<td>r</td>
<td>Types of values in the object.</td>
</tr>
<tr>
<td>591</td>
<td>UnicodeStrings</td>
<td>rw</td>
<td>Named access to Unicode string values</td>
</tr>
<tr>
<td>588</td>
<td>UnquotedMemberNames</td>
<td>rw</td>
<td>Should member names be written unquoted or quoted in JSON</td>
</tr>
</tbody>
</table>

16.16.4 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 (567) 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 (587), GetJSONObjectInstanceType (551)

16.16.5 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 (582) instance.

See also: TJSONObject.Clear (587), TJSONObject.Create (583)

16.16.6 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 (568)

16.16.7 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 (572)
16.16.8  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 (554).

See also: TJSONEnum (554), TJSONData.GetEnumerator (569)

16.16.9  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 (548)

16.16.10  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 (573) 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 (587), IndexOfName (585)

16.16.11  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 (573) 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 (585)
16.16.12 TJSONObject.Find

Synopsis: Find an element by name.

Declaration:  
function Find(const AName: string) : TJSONData; Overload  
function Find(const AName: string; AType: TJSONtype) : TJSONData  
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 (585), IndexOfName (585)

16.16.13 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 exists with the correct type exists, the ADefaultValue 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 (590), Int64s (590), Booleans (592), Strings (592), Floats (590), Arrays (592), and Objects (593) will raise an exception if the name is not found. The Get function does not raise an exception.

See also: Integers (590), Int64s (590), Booleans (592), Strings (592), Floats (590), Arrays (592), Objects (593)
16.16.14 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 (573) returns 0.

See also: Delete (587), Extract (588)

16.16.15 TJSONObject.Add

Synopsis: Add a name, value to the object

Declaration: function Add(const AName: TJSONStringType; AValue: TJSONData) : Integer
function Add(const AName: TJSONStringType; AValue: Boolean) : Integer
function Add(const AName: TJSONStringType; AValue: TJSONFloat) : Integer
function Add(const AName: TJSONStringType; const AValue: TJSONStringType) : Integer
function Add(const AName: string; AValue: TJSONUnicodeStringType) : Integer
function Add(const AName: TJSONStringType; Avalue: Int64) : Integer
function Add(const AName: TJSONStringType; Avalue: QWord) : Integer
function Add(const AName: TJSONStringType; Avalue: Integer) : Integer
function Add(const AName: TJSONStringType; AValue: TJSONArray) : Integer

Visibility: public

Description: Add adds the value AValue with name AName to the object. If the value is not a TJSONData (567) 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 (590), Int64s (590), Booleans (592), Strings (592), Floats (590), Arrays (592) and Objects (593) 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 (590), Int64s (590), Booleans (592), Strings (592), Floats (590), Arrays (592), Objects (593)

16.16.16 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 (567) 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 (587), Add (587), Extract (588), Exchange (582)

16.16.17 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 (587), Extract (588)

16.16.18 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 (543) 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 (543), Clear (543), Insert (543), Add (543)

16.16.19 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:

```json
{
   "name" : "Free Pascal",
   "type" : "Compiler"
}
```

When the value is True, JSON is written as:
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 JavaScript engine, the unquoted values are usually also accepted.

See also: TJSONData.CompressedJSON (572)

16.16.20 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 (589), Elements (589)

16.16.21 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 (554) exception is raised.

See also: Items (573), Names (589), Types (589), Integers (590), Int64s (590), Booleans (592), Strings (592), Floats (590), Arrays (592), Objects (593)

16.16.22 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 (554) exception is raised.

See also: Items (573), Names (589), Elements (589), Integers (590), Int64s (590), Booleans (592), Strings (592), Floats (590), Arrays (592), Nulls (590), Objects (593)
16.16.23 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 (554) exception is raised when reading. When writing, any existing value is replaced by a null value.

See also: TJSONData.Items (573), TJSONObject.Names (589), Elements (543), Integers (543), Int64s (543), Booleans (543), Strings (543), Floats (543), Arrays (543), Types (543), Objects (543)

16.16.24 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 (554) exception is raised when reading. When writing, any existing value is replaced by the specified floating-point value.

See also: Items (573), Names (589), Elements (589), Integers (590), Int64s (590), Booleans (592), Strings (592), Nulls (590), Arrays (592), Types (589), Objects (593)

16.16.25 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 (554) exception is raised when reading. When writing, any existing value is replaced by the specified integer value.

See also: TJSONData.Items (573), Names (589), Elements (589), Floats (590), Int64s (590), Booleans (592), Strings (592), Nulls (590), Arrays (592), Types (589), Objects (593)

16.16.26 TJSONObject.Int64s

Synopsis: Named access to int64 values

Declaration: Property Int64s[AName: string]: Int64

Visibility: public

590
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 (554) exception is raised when reading. When writing, any existing value is replaced by the specified int64 value.

See also: Items (573), Names (589), Elements (589), Floats (590), Integers (590), Booleans (592), Strings (592), Nulls (590), Arrays (592), Types (589), Objects (593)

16.16.27 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 (554) exception is raised when reading. When writing, any existing value is replaced by the specified QWord value.

See also: TJSONData.Items (573), Names (589), Elements (589), Floats (590), Integers (590), Booleans (592), Strings (592), Nulls (590), Arrays (592), Types (589), Objects (593)

16.16.28 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 (575) property when reading. Reading it is the equivalent to accessing

\[ \text{Items[Index].AsLargeInt} \]

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 (575), TJSONData.Items (573), TJSONData.IsNull (576), TJSONObject.Types (589), TJSONObject.Integers (590), TJSONObject.Floats (590), TJSONObject.Strings (592), TJSONObject.Booleans (592), TJSONObject.QWords (591)

16.16.29 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 (554) exception is raised when reading. When writing, any existing value is replaced by the specified UnicodeString value.

See also: Items (573), Names (589), Elements (589), Floats (590), Integers (590), Booleans (592), Int64s (590), Nulls (590), Arrays (592), Types (591), Objects (593), Strings (592)

16.16.30 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 (554) exception is raised when reading. When writing, any existing value is replaced by the specified string value.

See also: Items (573), Names (589), Elements (589), Floats (590), Integers (590), Booleans (592), Int64s (590), Nulls (590), Arrays (592), Types (589), Objects (593), UnicodeStrings (591)

16.16.31 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 (554) exception is raised when reading. When writing, any existing value is replaced by the specified boolean value.

See also: Items (573), Names (589), Elements (589), Floats (590), Integers (590), Strings (592), Int64s (590), Nulls (590), Arrays (592), Types (589), Objects (593)

16.16.32 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 (554) exception is raised when reading. When writing, any existing value is replaced by the specified JSON array.

See also: Items (573), Names (589), Elements (589), Floats (590), Integers (590), Strings (592), Int64s (590), Nulls (590), Booleans (592), Types (589), Objects (593)
16.16.33 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 (554) exception is raised when reading. When writing, any existing value is replaced by the specified JSON object.

See also: TJSONData.Items (573), Names (589), Elements (589), Floats (590), Integers (590), Strings (592), Int64s (590), Nulls (590), Booleans (592), Types (598), Arrays (592)

16.17 TJSONQWordNumber

16.17.1 Description

TJSONQWordNumber must be used whenever 64-bit unsigned integer data must be represented. For 32-bit integer data, TJSONIntegerNumber (579) must be used. For 64-bit signed integer data, TJSONInt64Number (578) must be used.

See also: TJSONFloatNumber (577), TJSONIntegerNumber (579), TJSONInt64Number (578)

16.17.2 Method overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>594</td>
<td>Clear</td>
<td>Clear value</td>
</tr>
<tr>
<td>594</td>
<td>Clone</td>
<td>Clone 64-bit unsigned integer value</td>
</tr>
<tr>
<td>593</td>
<td>Create</td>
<td>Create a new instance of 64-bit unsigned integer JSON data</td>
</tr>
<tr>
<td>593</td>
<td>NumberType</td>
<td>Kind of numerical data managed by this class</td>
</tr>
</tbody>
</table>

16.17.3 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 (580), TJSONInt64Number.Create (578), TJSONFloatNumber.Create (577)

16.17.4 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 (548), TJSONData.JSONtype (568)
16.17.5 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 (569)

16.17.6 TJSONQWordNumber.Clone

Synopsis: Clone 64-bit unsigned integer value

Declaration: function Clone : TJSONData; Override

Visibility: public

Description: Clone overrides TJSONData.Clone (572) and creates an instance of the same class with the 64-bit unsigned integer value.

See also: TJSONData.Clone (572)

16.18 TJSONString

16.18.1 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 (577), TJSONIntegerNumber (579), TJSONInt64Number (578), TJSONBoolean (566), TJSONNull (580), TJSONArray (555), TJSONObject (582)

16.18.2 Method overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>595</td>
<td>Clear</td>
<td>Clear value</td>
</tr>
<tr>
<td>595</td>
<td>Clone</td>
<td>Clone string value</td>
</tr>
<tr>
<td>594</td>
<td>Create</td>
<td>Create a new instance of string JSON data</td>
</tr>
<tr>
<td>595</td>
<td>JSONType</td>
<td>native JSON data type</td>
</tr>
</tbody>
</table>

16.18.3 TJSONString.Create

Synopsis: Create a new instance of string JSON data

Declaration: constructor Create(const AValue: TJSONStringType); Reintroduce
cTORuctor 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.
16.18.4 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 (568)

16.18.5 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 (569)

16.18.6 TJSONString.Clone

Synopsis: Clone string value

Declaration: function Clone: TJSONData; Override

Visibility: public

Description: Clone overrides TJSONData.Clone (572) and creates an instance of the same class with the same string value.

See also: TJSONData.Clone (572)
Chapter 17

Reference for unit ’fpmimetypes’

17.1 Used units

Table 17.1: Used units by unit ’fpmimetypes’

<table>
<thead>
<tr>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Classes</td>
<td>??</td>
</tr>
<tr>
<td>Contnrs</td>
<td>201</td>
</tr>
<tr>
<td>System</td>
<td>??</td>
</tr>
<tr>
<td>sysutils</td>
<td>??</td>
</tr>
</tbody>
</table>

17.2 Overview

The fpmimetypes unit contains a class which handles mapping of filename extensions to MIME (Multipurpose Internet Mail Extensions) types. The TFPMimeTypes (597) class handles this mapping. A global instance of this class is available through the MimeTypes (596) function. The list of known mime types can be instantiated through the LoadKnownTypes (598) method, or a file in the standard mime.types format can be loaded through the LoadFromFile (599) method.

17.3 Procedures and functions

17.3.1 MimeTypes

Synopsis: Global TFPMimeTypes instance

Declaration: function MimeTypes : TFPMimeTypes

Visibility: default

Description: MimeTypes returns a global instance of the TFPMimeTypes (597) class. It is not initialized with a list of extensions, so this instance must still be explicitly initialized with TFPMimeTypes.LoadKnownTypes (598). 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 (597), TFPMimeTypes.LoadKnownTypes (598)
17.4 TFP_mimeTypes

17.4.1 Description

TFP_mimeTypes manages a list of MIME types. The list of types can be initialized with the OS list of known MIME types through the LoadKnownTypes (598) method, or a file in the standard mime.types format can be loaded through the LoadFromFile (599) method. The associated mime type of a file extension can be retrieved with TFP_mimeTypes.GetMimeType (599).

See also: TFP_mimeTypes.LoadKnownTypes (598), TFP_mimeTypes.LoadFromFile (599), TFP_mimeTypes.GetMimeType (599)

17.4.2 Method overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>599</td>
<td>AddType</td>
<td>Add a MIME type to the list.</td>
</tr>
<tr>
<td>598</td>
<td>Clear</td>
<td>Clear the list of known MIME types</td>
</tr>
<tr>
<td>597</td>
<td>Create</td>
<td>Create a new instance of the TFP_mimeTypes class</td>
</tr>
<tr>
<td>597</td>
<td>Destroy</td>
<td>Remove instance from memory</td>
</tr>
<tr>
<td>600</td>
<td>GetKnownExtensions</td>
<td>Get a list of all known extensions</td>
</tr>
<tr>
<td>600</td>
<td>GetKnownMimeTypes</td>
<td>Get a list of all known MIME types</td>
</tr>
<tr>
<td>599</td>
<td>GetMimeExtensions</td>
<td>Get the extensions associated with a MIME type</td>
</tr>
<tr>
<td>599</td>
<td>GetMimeType</td>
<td>Get MIME type of an extension</td>
</tr>
<tr>
<td>598</td>
<td>GetNextExtension</td>
<td>Extract an extension from an extension list as returned by GetMimeExtensions</td>
</tr>
<tr>
<td>599</td>
<td>LoadFromFile</td>
<td>Load mime types from a file in mime.types file format</td>
</tr>
<tr>
<td>598</td>
<td>LoadFromStream</td>
<td>Load mime types from a stream in mime.types file format</td>
</tr>
<tr>
<td>597</td>
<td>LoadKnownTypes</td>
<td>Queries the OS for a list of known MIME types</td>
</tr>
</tbody>
</table>

17.4.3 TFP_mimeTypes.Create

Synopsis: Create a new instance of the TFP_mimeTypes class

Declaration: constructor Create(AOwner: TComponent); Override

Visibility: public

Description: The Create method sets up the necessary internal structures.

See also: TFP_mimeTypes.Destroy (597)

17.4.4 TFP_mimeTypes.Destroy

Synopsis: Remove instance from memory

Declaration: destructor Destroy; Override

Visibility: public

Description: Destroy destroys the TFP_mimeTypes instance and removes it from memory.

See also: TFP_mimeTypes.Create (597)
17.4.5  TFPMimeType.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: TFPMimeType.LoadKnownTypes (598)

17.4.6  TFPMimeType.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: TFPMimeType.LoadFromFile (599)

17.4.7  TFPMimeType.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: TFPMimeType.GetMimeExtensions (599)

17.4.8  TFPMimeType.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: TFPMimeType.LoadFromFile (599)
17.4.9 TFPMimeType.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: TFPMimeType.LoadFromStream (598)

17.4.10 TFPMimeType.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: TFPMimeType.GetMimeExtensions (599), TFPMimeType.GetMimeType (599)

17.4.11 TFPMimeType.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 insenitively.

See also: TFPMimeType.GetMimeType (599)

17.4.12 TFPMimeType.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: TFPMimeType.GetMimeExtensions (599)
17.4.13 TFPMimeType.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: TFPMimeType.GetKnownExtensions (600)

17.4.14 TFPMimeType.GetKnownExtensions

Synopsis: Get a list 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: TFPMimeType.GetKnownMimeTypes (600)

17.5 TMimeType

17.5.1 Description

TMimeType is a helper class which stores a MIME type and its associated extensions. It should not be necessary to create instances of this class manually, the creation is handled entirely through the TFPMimeType (597) class.

See also: TMimeType.MimeType (601), TMimeType.Extensions (601), TFPMimeType (597)

17.5.2 Method overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>601</td>
<td>Create</td>
<td>Create a new instance of TMimeType</td>
</tr>
<tr>
<td>601</td>
<td>MergeExtensions</td>
<td>Merge extensions in the list of extensions</td>
</tr>
</tbody>
</table>

17.5.3 Property overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Properties</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>601</td>
<td>Extensions</td>
<td>rw</td>
<td>Semicolon-separated list of extensions associated with MimeType</td>
</tr>
<tr>
<td>601</td>
<td>MimeType</td>
<td>rw</td>
<td>Mime type name</td>
</tr>
</tbody>
</table>
17.5.4  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 (601) TMimeType.Extensions (601) properties to aMimeType and aExtensions.

See also: TMimeType.MimeType (601), TMimeType.Extensions (601)

17.5.5  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 (601) in such a way that there are no duplicates.

See also: TMimeType.Extensions (601)

17.5.6  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 (601)

17.5.7  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 (601)

See also: TMimeType.MimeType (601)
Chapter 18

Reference for unit ’fptimer’

18.1 Used units

Table 18.1: Used units by unit ’fptimer’

<table>
<thead>
<tr>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Classes</td>
<td>??</td>
</tr>
<tr>
<td>System</td>
<td>??</td>
</tr>
</tbody>
</table>

18.2 Overview

The fpTimer unit implements a timer class TFPTimer (604) which can be used on all supported platforms. The timer class uses a driver class TFPTimerDriver (606) 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 (602) variable to the class pointer of the driver class. The driver class should descend from TFPTimerDriver (606).

18.3 Constants, types and variables

18.3.1 Types

TFPTimerDriverClass = Class of TFPTimerDriver

TFPTimerDriverClass is the class pointer of TFPTimerDriver (606) it exists mainly for the purpose of being able to set DefaultTimerDriverClass (602), so a custom timer driver can be used for the timer instances.

18.3.2 Variables

DefaultTimerDriverClass : TFPTimerDriverClass = Nil
DefaultTimerDriverClass contains the TFPTimerDriver (606) class pointer that should be used when a new instance of TFPCustomTimer (603) 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.

18.4 TFPCustomTimer

18.4.1 Description

TFPCustomTimer is the timer class containing the timer's implementation. It relies on an extra driver instance (of type TFPTimerDriver (606)) to do the actual work.

TFPCustomTimer publishes no events or properties, so it is unsuitable for handling in an IDE. The TFP Timer (604) descendent class publishes all needed events of TFPCustomTimer.

See also: TFP TimerDriver (606), TFP Timer (604)

18.4.2 Method overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>603</td>
<td>Create</td>
<td>Create a new timer</td>
</tr>
<tr>
<td>603</td>
<td>Destroy</td>
<td>Release a timer instance from memory</td>
</tr>
<tr>
<td>604</td>
<td>StartTimer</td>
<td>Start the timer</td>
</tr>
<tr>
<td>604</td>
<td>StopTimer</td>
<td>Stop the timer</td>
</tr>
</tbody>
</table>

18.4.3 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 (603)

18.4.4 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 (603)
18.4.5 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 (604) event is called.

See also: StopTimer (604), Enabled (604), OnTimer (605)

18.4.6 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 (604), Enabled (604), OnTimer (605)

18.5 TFPTimer

18.5.1 Description

TFPTimer implements no new events or properties, but merely publishes events and properties already implemented in TFPCustomTimer (603): Enabled (604), OnTimer (605) and Interval (605).

The TFPTimer class is suitable for use in an IDE.

See also: TFPCustomTimer (603), Enabled (604), OnTimer (605), Interval (605)

18.5.2 Property overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Properties</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>604</td>
<td>Enabled</td>
<td></td>
<td>Start or stop the timer</td>
</tr>
<tr>
<td>605</td>
<td>Interval</td>
<td></td>
<td>Timer tick interval in milliseconds.</td>
</tr>
<tr>
<td>605</td>
<td>OnStartTimer</td>
<td></td>
<td></td>
</tr>
<tr>
<td>606</td>
<td>OnStopTimer</td>
<td></td>
<td></td>
</tr>
<tr>
<td>605</td>
<td>OnTimer</td>
<td></td>
<td>Event called on each timer tick.</td>
</tr>
<tr>
<td>605</td>
<td>UseTimerThread</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

18.5.3 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 (604)), setting it to False will stop the timer (calling StopTimer (604)).

See also: StartTimer (604), StopTimer (604), OnTimer (605), Interval (605)

18.5.4 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 (605) 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 (605), Enabled (604)

18.5.5 TFPTimer.UseTimerThread

Declaration: Property UseTimerThread :

Visibility: published

Access:

18.5.6 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 (605), Enabled (604)

18.5.7 TFPTimer.OnStartTimer

Declaration: Property OnStartTimer :

Visibility: published

Access:
### 18.5.8 TFPTimer.OnStopTimer

**Declaration:**

Property OnStopTimer:

- **Visibility:** published
- **Access:**

### 18.6 TFPTimerDriver

#### 18.6.1 Description

TFPTimerDriver is the abstract timer driver class: it simply provides an interface for the TFP-CustomTimer (603) class to use.

The fpTimer unit implements a descendant of this class which implements the default timer mechanism.

**See also:** TFPCustomTimer (603), DefaultTimerDriverClass (602)

#### 18.6.2 Method overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>606</td>
<td>Create</td>
<td>Creates a new driver instance</td>
</tr>
<tr>
<td>606</td>
<td>StartTimer</td>
<td>Start the timer</td>
</tr>
<tr>
<td>607</td>
<td>StopTimer</td>
<td>Stop the timer</td>
</tr>
</tbody>
</table>

#### 18.6.3 Property overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Properties</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>607</td>
<td>Timer</td>
<td>r</td>
<td>Timer tick</td>
</tr>
<tr>
<td>607</td>
<td>TimerStarted</td>
<td>r</td>
<td>True when the timer driver has called its StartTimer method</td>
</tr>
</tbody>
</table>

#### 18.6.4 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 (607)) a reference to the ATimer instance which created the driver instance.

**See also:** Timer (607), TFPTimer (604)

#### 18.6.5 TFPTimerDriver.StartTimer

**Synopsis:** Start the timer

**Declaration:**

`procedure StartTimer; Virtual; Abstract`

- **Visibility:** public
Description: StartTimer is called by TFPCustomTimer.StartTimer (604). It should be overridden by descendents of TFPTimerDriver to actually start the timer.

See also: TFPCustomTimer.StartTimer (604), TFPTimerDriver.StopTimer (607)

18.6.6 TFPTimerDriver.StopTimer

Synopsis: Stop the timer

Declaration: procedure StopTimer; Virtual; Abstract

Visibility: public

Description: StopTimer is called by TFPCustomTimer.StopTimer (604). It should be overridden by descendents of TFPTimerDriver to actually stop the timer.

See also: TFPCustomTimer.StopTimer (604), TFPTimerDriver.StartTimer (606)

18.6.7 TFPTimerDriver.Timer

Synopsis: Timer tick

Declaration: Property Timer : TFPCustomTimer

Visibility: public

Access: Read

Description: Timer calls the TFPCustomTimer (603) timer event. Descendents of TFPTimerDriver should call Timer whenever a timer tick occurs.

See also: TFPTimer.OnTimer (605), TFPTimerDriver.StartTimer (606), TFPTimerDriver.StopTimer (607)

18.6.8 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 19

Reference for unit ’gettext’

19.1 Used units

Table 19.1: Used units by unit ’gettext’

<table>
<thead>
<tr>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Classes</td>
<td>??</td>
</tr>
<tr>
<td>System</td>
<td>??</td>
</tr>
<tr>
<td>sysutils</td>
<td>??</td>
</tr>
</tbody>
</table>

19.2 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 (611)) 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.

19.3 Constants, types and variables

19.3.1 Constants

MOFileHeaderMagic = $950412DE

This constant is found as the first integer in a .mo

19.3.2 Types

PLongWordArray = ^TLongWordArray

Pointer to a TLongWordArray (609) array.

PMOStringTable = ^TMOStringTable
Pointer to a TMOStringTable (609) array.

PChar = ^TPChar

Pointer to a TPCharArray (609) array.

TLongWordArray = Array[0..(1 shl 30) div SizeOf(LongWord)] of LongWord

TLongWordArray is an array used to define the PLongWordArray (608) 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..(1 shl 30) div SizeOf(TMOStringInfo)] of TMOStringInfo

TMOStringTable is an array type containing TMOStringInfo (610) records. It should never be used directly, as it would occupy too much memory.

TPCharArray = Array[0..(1 shl 30) div SizeOf(PChar)] of PChar

TPCharArray is an array used to define the PPCharArray (609) 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.

19.4 Procedures and functions

19.4.1 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.

19.4.2 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 (611) 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 (610), TMOFile (611)
19.4.3 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 (609), 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 (609), TMOFile (611)

19.5 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.

19.6 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 TMOFile-Header (610) record at the start of the file.

19.7 EMOFileError

19.7.1 Description

EMOFileError is raised in case an TMOFile (611) instance is created with an invalid .mo.

See also: TMOFile (611)
19.8 TMOFile

19.8.1 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.

19.8.2 Method overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>611</td>
<td>Create</td>
<td>Create a new instance of the TMOFile class.</td>
</tr>
<tr>
<td>611</td>
<td>Destroy</td>
<td>Removes the TMOFile instance from memory</td>
</tr>
<tr>
<td>611</td>
<td>Translate</td>
<td>Translate a string</td>
</tr>
</tbody>
</table>

19.8.3 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 (610) structure, then an EMOFileError (610) exception is raised.

See also: TMOFile.Destroy (611)

19.8.4 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 (611)

19.8.5 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 20

Reference for unit ’IBConnection’

20.1 Used units

Table 20.1: Used units by unit ’IBConnection’

<table>
<thead>
<tr>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>BufDataset</td>
<td>131</td>
</tr>
<tr>
<td>Classes</td>
<td>??</td>
</tr>
<tr>
<td>DB</td>
<td>319</td>
</tr>
<tr>
<td>dbconst</td>
<td>??</td>
</tr>
<tr>
<td>ibase60dyn</td>
<td>??</td>
</tr>
<tr>
<td>SQLDB</td>
<td>786</td>
</tr>
<tr>
<td>System</td>
<td>??</td>
</tr>
<tr>
<td>sysutils</td>
<td>??</td>
</tr>
</tbody>
</table>

20.2 Constants, types and variables

20.2.1 Constants

`DEFDIALECT = 3`

Default dialect that will be used when connecting to databases. See TIBConnection.Dialect (618) for more details on dialects.

`MAXBLOBSEGMENTSIZE = 65535`

Maximum size to use when fetching blob segments

20.2.2 Types

`TStatusVector = Array[0..19] of ISC_STATUS`

`TStatusVector` is the (opaque) type of the Interbase internal status vector.
20.3 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.

20.4 EIBDatabaseError

20.4.1 Description

Firebird/Interbase database error, a descendant of db.EDatabaseError (613).

See also: db.EDatabaseError (613)

20.4.2 Property overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Properties</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>614</td>
<td>GDSErrorCode</td>
<td>r</td>
<td>Firebird/Interbase GDS error code.</td>
</tr>
<tr>
<td>614</td>
<td>StatusVector</td>
<td>rw</td>
<td>Low-level IB API status vector</td>
</tr>
</tbody>
</table>

20.4.3 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.

20.4.4 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.
20.5  TIBConnection

20.5.1  Description

TIBConnection is a descendant of TSQLConnection (613) 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:

```delphi
MyQuery.FieldByName('full_name').ProviderFlags:=[];
```

See also: TSQLConnection (613)
20.5.2 Method overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>616</td>
<td>Create</td>
<td>Creates a TIBConnection object</td>
</tr>
<tr>
<td>617</td>
<td>CreateDB</td>
<td>Creates a database on disk</td>
</tr>
<tr>
<td>617</td>
<td>DropDB</td>
<td>Deletes a database from disk</td>
</tr>
<tr>
<td>616</td>
<td>GetConnectionInfo</td>
<td>Return some information about the connection</td>
</tr>
</tbody>
</table>

20.5.3 Property overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Properties</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>617</td>
<td>BlobSegmentSize</td>
<td>rw</td>
<td>Write this amount of bytes per BLOB segment</td>
</tr>
<tr>
<td>619</td>
<td>CheckTransactionParams</td>
<td>rw</td>
<td>Let StartTransaction check transaction parameters</td>
</tr>
<tr>
<td>618</td>
<td>DatabaseName</td>
<td>rw</td>
<td>Name of the database to connect to</td>
</tr>
<tr>
<td>618</td>
<td>Dialect</td>
<td>rws</td>
<td>Database dialect</td>
</tr>
<tr>
<td>619</td>
<td>KeepConnection</td>
<td>rws</td>
<td>Keep open connection after first query</td>
</tr>
<tr>
<td>619</td>
<td>LoginPrompt</td>
<td>rws</td>
<td>Switch for showing custom login prompt</td>
</tr>
<tr>
<td>618</td>
<td>ODSMajorVersion</td>
<td>r</td>
<td>Database On-Disk Structure major version</td>
</tr>
<tr>
<td>620</td>
<td>OnLogin</td>
<td>s</td>
<td>Event triggered when a login prompt needs to be shown.</td>
</tr>
<tr>
<td>619</td>
<td>Params</td>
<td>rw</td>
<td>Firebird/Interbase specific parameters</td>
</tr>
<tr>
<td>620</td>
<td>Port</td>
<td>s</td>
<td>Port at which the server listens</td>
</tr>
<tr>
<td>620</td>
<td>UseConnectionCharSetIfNone</td>
<td>rw</td>
<td>For string/blob fields with codepage none, use the connection character set when copying data</td>
</tr>
<tr>
<td>620</td>
<td>WireCompression</td>
<td>rw</td>
<td>Use wire compression when communicating with the server</td>
</tr>
</tbody>
</table>

20.5.4 TIBConnection.Create

Synopsis: Creates a TIBConnection object

Declaration: constructor Create(AOwner: TComponent); Override

Visibility: public

Description: Creates a TIBConnection object

20.5.5 TIBConnection.GetConnectionInfo

Synopsis: Return some information about the connection

Declaration: function GetConnectionInfo(InfoType: TConnInfoType) : string; Override

Visibility: public

Description: GetConnectionInfo overrides TSQLConnection.GetConnectionInfo (817) to return the relevant information for the Interbase/Firebird connection.

See also: TSQLConnection.GetConnectionInfo (817), TConnInfoType (796)
20.5.6 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 (613) (specifically, PAGE_SIZE) and TSQLConnection.CharSet (613) 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 (613), TSQLConnection.UserName (613), TSQLConnection.Password (613).
- If creating a database using Firebird embedded, make sure the embedded library is loaded, the TSQLConnection.HostName (613) property is empty, and set the TSQLConnection.UserName (613) to e.g. 'SYSDBA'. See TIBConnection: Firebird/Interbase specific TSQLConnection (613) for details on loading the embedded database library.

See also: TSQLConnection.Params (613), TSQLConnection.DropDB (613), TIBConnection (615)

20.5.7 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 (615).

- 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 (613), TSQLConnection.UserName (613), TSQLConnection.Password (613).
- When using Firebird embedded, make sure the embedded connection library is loaded, the TSQLConnection.HostName (613) property is empty, and set the TSQLConnection.UserName (613) to e.g. 'SYSDBA'. See TIBConnection: Firebird/Interbase specific TSQLConnection (613) for more details on loading the embedded library.

See also: TSQLConnection.CreateDB (613), TSQLConnection.HostName (613), TSQLConnection.UserName (613), TSQLConnection.Password (613)

20.5.8 TIBConnection.BlobSegmentSize

Synopsis: Write this amount of bytes per BLOB segment

Declaration: Property BlobSegmentSize : Word; deprecated;

Visibility: public

Access: Read, Write
CHAPTER 20. REFERENCE FOR UNIT 'IBCONNECTION'

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.

20.5.9 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.

20.5.10 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).

20.5.11 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.
20.5.12 **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.

20.5.13 **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.

20.5.14 **TIBConnection>LoginPrompt**

**Synopsis:** Switch for showing custom login prompt

**Declaration:**
```
Property LoginPrompt :
```

**Visibility:** published

**Access:**

**Description:** If true, the `OnLogin` (613) event will fire, allowing you to handle supplying of credentials yourself.

See also: TSQLConnection.OnLogin (613)

20.5.15 **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 (824)

619
20.5.16  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 (613) property is True, after the BeforeConnect (365) event, but before the connection is actually established.

See also: #fcl.db.TCustomConnection.BeforeConnect (365), TSQLConnection.LoginPrompt (613), #fcl.db.TCustomConnection.Open (362), TSQLConnection.OnLogin (613)

20.5.17  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 activating the connection.

20.5.18  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 (820)

20.5.19  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 activating the connection.
20.6 TIBConnectionDef

20.6.1 Description
Child of TConnectionDef (613) 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 (613)

20.6.2 Method overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>621</td>
<td>ConnectionClass</td>
<td>Firebird/Interbase child of ConnectionClass (803)</td>
</tr>
<tr>
<td>622</td>
<td>DefaultLibraryName</td>
<td>Default name of the firebird client library</td>
</tr>
<tr>
<td>621</td>
<td>Description</td>
<td>Description for the Firebird/Interbase child of #fcl.sqldb.TConnectionDef.ConnectionClass (803)</td>
</tr>
<tr>
<td>622</td>
<td>LoadedLibraryName</td>
<td>Actually loaded library name</td>
</tr>
<tr>
<td>622</td>
<td>LoadFunction</td>
<td>Return Function to call when loading firebird support</td>
</tr>
<tr>
<td>621</td>
<td>TypeName</td>
<td>Firebird/Interbase child of TConnectionDef.TypeName (613)</td>
</tr>
<tr>
<td>622</td>
<td>UnLoadFunction</td>
<td>Return Function to call when unloading firebird support</td>
</tr>
</tbody>
</table>

20.6.3 TIBConnectionDef.TypeName

Synopsis: Firebird/Interbase child of TConnectionDef.TypeName (613)

Declaration: class function TypeName : string; Override

Visibility: default

See also: TConnectionDef.TypeName (613), TIBConnection (615)

20.6.4 TIBConnectionDef.ConnectionClass

Synopsis: Firebird/Interbase child of ConnectionClass (803)

Declaration: class function ConnectionClass : TSQLConnectionClass; Override

Visibility: default

See also: TConnectionDef.ConnectionClass (613), TIBConnection (615)

20.6.5 TIBConnectionDef.Description

Synopsis: Description for the Firebird/Interbase child of #fcl.sqldb.TConnectionDef.ConnectionClass (803)

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 (803), TIBConnection (615)
20.6.6 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.

20.6.7 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.

20.6.8 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.

20.6.9 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 (622)

20.7 TIBCursor
20.7.1 Description
A cursor that keeps track of where you are in a Firebird/Interbase dataset. It is a descendent of TSQLCursor (613).
See also: TSQLCursor (613), TIBConnection (615)
20.8  TIBTrans

20.8.1  Description
Firebird/Interbase database transaction object. Descendant of TSQLHandle (613).

See also: TSQLHandle (613), TIBConnection (615)
Chapter 21

Reference for unit ’idea’

21.1 Used units

Table 21.1: Used units by unit ’idea’

<table>
<thead>
<tr>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Classes</td>
<td>??</td>
</tr>
<tr>
<td>System</td>
<td>??</td>
</tr>
<tr>
<td>sysutils</td>
<td>??</td>
</tr>
</tbody>
</table>

21.2 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.

21.3 Constants, types and variables

21.3.1 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
21.3.2 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 (626) or DeKeyIDEA (625) 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.

21.4 Procedures and functions

21.4.1 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 (626), DeKeyIDEA (625), TIDEAEncryptStream (628), TIDEADecryptStream (626)

21.4.2 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 \( \text{dk} \). 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 (626), CipherIdea (625)

21.4.3 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 (625), CipherIdea (625)

21.5 EIDEAError

21.5.1 Description

EIDEAError is used to signal errors in the IDEA encryption decryption streams.

21.6 TIDEADecryptStream

21.6.1 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 (625) 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 (628) encryption stream.

See also: TIDEAEncryptStream (628), TIDEAStream.Create (630), CipherIDEA (625)

21.6.2 Method overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>627</td>
<td>Create</td>
<td>Constructor to create a new TIDEADecryptStream instance</td>
</tr>
<tr>
<td>627</td>
<td>Read</td>
<td>Reads data from the stream, decrypting it as needed</td>
</tr>
<tr>
<td>627</td>
<td>Seek</td>
<td>Set position on the stream</td>
</tr>
</tbody>
</table>
### 21.6.3 TIDEADeCryptStream.Create

**Synopsis:** Constructor to create a new TIDEADecryptStream instance

**Declaration:**
```pascal
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 (625), which is then passed on to the inherited constructor TIDEAStream.Create (630). It is an easy-access function which introduces no new functionality.

The string is truncated at the maximum length of the TIdeaCryptKey (625) structure, so it makes no sense to provide a string with length longer than this structure.

See also: TIdeaCryptKey (625), TIDEAStream.Create (630), TIDEAEnCryptStream.Create (628)

### 21.6.4 TIDEADeCryptStream.Read

**Synopsis:** Reads data from the stream, decrypting it as needed

**Declaration:**
```pascal
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 (627), TIDEAEncryptStream (628)

### 21.6.5 TIDEADeCryptStream.Seek

**Synopsis:** Set position on the stream

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

**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 (626) exception is raised if the stream does not allow the requested seek operation.

See also: Read (627)
21.7 TIDEAEncryptStream

21.7.1 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 (625) 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 (626) decryption stream.

See also: TIDEADecryptStream (626), TIDEAStream.Create (630), CipherIDEA (625)

21.7.2 Method overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>628</td>
<td>Create</td>
<td>Constructor to create a new TIDEAEncryptStream instance</td>
</tr>
<tr>
<td>628</td>
<td>Destroy</td>
<td>Flush data buffers and free the stream instance.</td>
</tr>
<tr>
<td>629</td>
<td>Flush</td>
<td>Write remaining bytes from the stream</td>
</tr>
<tr>
<td>629</td>
<td>Seek</td>
<td>Set stream position</td>
</tr>
<tr>
<td>629</td>
<td>Write</td>
<td>Write bytes to the stream to be encrypted</td>
</tr>
</tbody>
</table>

21.7.3 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 (625), which is then passed on to the inherited constructor TIDEAStream.Create (630). It is an easy-access function which introduces no new functionality.

The string is truncated at the maximum length of the TIdEAEncryptKey (625) structure, so it makes no sense to provide a string with length longer than this structure.

See also: TIdEAEncryptKey (625), TIDEAStream.Create (630), TIDEADecryptStream.Create (627)

21.7.4 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 (630)
21.7.5 TIDEAEncryptStream.Write

Synopsis: Write bytes to the stream to be encrypted

Declaration: function Write(const Buffer: Word; 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 (627)

21.7.6 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 (626) will be raised unless called with 0 and soFromCurrent as arguments.

See also: Write (629), EIDEAError (626)

21.7.7 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 (629)

21.8 TIDEAStream

21.8.1 Description

Do not create instances of TIDEAStream directly. It implements no useful functionality: it serves as a common ancestor of the TIDEAEncryptStream (628) and TIDEADeCryptStream (626), 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 (628), TIDEADeCryptStream (626)
21.8.2 **Method overview**

<table>
<thead>
<tr>
<th>Page</th>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>630</td>
<td>Create</td>
<td>Creates a new instance of the <strong>TIDEAStream</strong> class</td>
</tr>
</tbody>
</table>

21.8.3 **Property overview**

<table>
<thead>
<tr>
<th>Page</th>
<th>Properties</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>630</td>
<td>Key</td>
<td>r</td>
<td>Key used when encrypting/decrypting</td>
</tr>
</tbody>
</table>

21.8.4 **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 (628), TIDEADecryptStream (626)

21.8.5 **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 (625)
Chapter 22

Reference for unit 'inicol'

22.1 Used units

Table 22.1: Used units by unit 'inicol'

<table>
<thead>
<tr>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Classes</td>
<td>??</td>
</tr>
<tr>
<td>IniFiles</td>
<td>641</td>
</tr>
<tr>
<td>System</td>
<td>??</td>
</tr>
<tr>
<td>sysutils</td>
<td>??</td>
</tr>
</tbody>
</table>

22.2 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 (643) class for this.

22.3 Constants, types and variables

22.3.1 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.
22.4 EIniCol

22.4.1 Description
EIniCol is used to report error conditions in the load and save methods of TIniCollection (632).

22.5 TIniCollection

22.5.1 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 (636).

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 (634), TIniCollection.LoadFromIni (634), TIniCollection.SaveToIni (633), TIniCollection.SaveToFile (633)

22.5.2 Method overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>632</td>
<td>Load</td>
<td>Loads the collection from the default filename.</td>
</tr>
<tr>
<td>634</td>
<td>LoadFromFile</td>
<td>Load collection from file.</td>
</tr>
<tr>
<td>634</td>
<td>LoadFromIni</td>
<td>Load collection from a file in .ini file format.</td>
</tr>
<tr>
<td>633</td>
<td>Save</td>
<td>Save the collection to the default filename.</td>
</tr>
<tr>
<td>633</td>
<td>SaveToFile</td>
<td>Save collection to a file in .ini file format</td>
</tr>
<tr>
<td>633</td>
<td>SaveToIni</td>
<td>Save the collection to a TCustomIniFile descendent</td>
</tr>
</tbody>
</table>

22.5.3 Property overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Properties</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>635</td>
<td>FileName</td>
<td>rw</td>
<td>Filename of the collection</td>
</tr>
<tr>
<td>635</td>
<td>GlobalSection</td>
<td>rw</td>
<td>Name of the global section</td>
</tr>
<tr>
<td>634</td>
<td>Prefix</td>
<td>r</td>
<td>Prefix used in global section</td>
</tr>
<tr>
<td>635</td>
<td>SectionPrefix</td>
<td>r</td>
<td>Prefix string for section names</td>
</tr>
</tbody>
</table>

22.5.4 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 (635) property. It calls the LoadFromFile (634) method to do this.

Errors: If the collection was not loaded or saved to file before this call, an EIniCol exception will be raised.
22.5.5  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 (635) property, using GlobalSection (635) as the section. It calls the SaveToFile (633) 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 (633), TIniCollection.SaveToIni (633), TIniCollection.Load (632), FileName (635)

22.5.6  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 (634) property for the key. After this it calls the SaveToIni (636) method of all TIniCollectionItem (636) 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 (635) property contains the value of Section, it will be used in the Save (633) method.

See also: TIniCollectionItem.SaveToIni (636), TIniCollection.Save (633)

22.5.7  TIniCollection.SaveToFile

Synopsis: Save collection to a file in .ini file format

Declaration: procedure SaveToFile(AFileName: string; Section: string)

Visibility: public
CHAPTER 22. REFERENCE FOR UNIT 'INICOL'

**22.5.8 TIniCollection.LoadFromIni**

**Synopsis:** Load collection from a file in .ini file format.

**Declaration:**

```pascal
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 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 method description.

**See also:** TIniCollection.LoadFromIni, TIniCollectionItem.LoadFromIni, TIniCollection.SaveToIni

**22.5.9 TIniCollection.LoadFromFile**

**Synopsis:** Load collection from file.

**Declaration:**

```pascal
procedure LoadFromFile(AFileName: string; Section: string)
```

**Visibility:** public

**Description:** LoadFromFile creates a TMemIniFile instance using AFileName as the filename. It calls LoadFromIni using this instance and Section as the parameters.

**See also:** TIniCollection.LoadFromIni, TIniCollection.Load, TIniCollection.SaveToIni, TIniCollection.SaveToFile

**22.5.10 TIniCollection.Prefix**

**Synopsis:** Prefix used in global section

**Declaration:**

```pascal
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:
A descendent of TIniCollection should set the value of this property, it cannot be empty.

See also: TIniCollection.SectionPrefix (635), TIniCollection.GlobalSection (635)

**22.5.11 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 (637) 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 (635), TIniCollectionItem.SectionName (637)

**22.5.12 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 (634) or SaveToFile (633) operation. It is used in the Load (632) or Save (633) calls.

See also: Save (633), LoadFromFile (634), SaveToFile (633), Load (632)

**22.5.13 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 (634) or SaveToIni (633) calls. It's used in the Load (632) or Save (633) calls.

See also: Save (633), LoadFromFile (634), SaveToFile (633), Load (632)
22.6  TIniCollectionItem

22.6.1  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 (636) and LoadFromIni (636) methods should be overridden. They should implement the actual loading and saving. The loading and saving is always initiated by the methods in TIniCollection (632), TIniCollection.LoadFromIni (634) and TIniCollection.SaveToIni (633) respectively.

See also: TIniCollection (632), TIniCollectionItem.SaveToIni (636), TIniCollectionItem.LoadFromIni (636), TIniCollection.LoadFromIni (634), TIniCollection.SaveToIni (633)

22.6.2  Method overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>637</td>
<td>LoadFromFile</td>
<td>Load item from a file</td>
</tr>
<tr>
<td>636</td>
<td>LoadFromIni</td>
<td>Method called when the item must be loaded</td>
</tr>
<tr>
<td>637</td>
<td>SaveToFile</td>
<td>Save item to a file</td>
</tr>
<tr>
<td>636</td>
<td>SaveToIni</td>
<td>Method called when the item must be saved</td>
</tr>
</tbody>
</table>

22.6.3  Property overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Properties</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>637</td>
<td>SectionName</td>
<td>rw</td>
<td>Default section name</td>
</tr>
</tbody>
</table>

22.6.4  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 (633) 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 (633), LoadFromIni (636), SaveToFile (637), LoadFromFile (637)

22.6.5  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 (634) 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 (634), SaveToIni (636), LoadFromFile (637), SaveToFile (637)

22.6.6 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 (636) 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 (636), LoadFromFile (637)

22.6.7 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 (636) to load the item from the indicated file in .ini format from the section Section.

Errors: None.

See also: SaveToFile (637), LoadFromIni (636)

22.6.8 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 (637), LoadFromIni (636)
22.7 **TNamedIniCollection**

### 22.7.1 Description

TNamedIniCollection is the collection to go with the TNamedIniCollectionItem (639) item class. It provides some functions to look for items based on the UserData (638) or based on the Name (638).

See also: TNamedIniCollectionItem (639), IndexOfUserData (638), IndexOfName (638)

### 22.7.2 Method overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>639</td>
<td>FindByName</td>
<td>Return the item based on its name</td>
</tr>
<tr>
<td>639</td>
<td>FindByUserData</td>
<td>Return the item based on its UserData</td>
</tr>
<tr>
<td>638</td>
<td>IndexOfName</td>
<td>Search for an item, based on its name, and return its position</td>
</tr>
<tr>
<td>638</td>
<td>IndexOfUserData</td>
<td>Search for an item based on its UserData property</td>
</tr>
</tbody>
</table>

### 22.7.3 Property overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Properties</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>639</td>
<td>NamedItems</td>
<td>rw</td>
<td>Indexed access to the TNamedIniCollectionItem items</td>
</tr>
</tbody>
</table>

### 22.7.4 TNamedIniCollection.IndexOfUserData

**Synopsis:** Search for an item based on its UserData property

**Declaration:**

```pascal
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 (638) property. If no such item exists, -1 is returned.

Note that the (linear) search starts at the last element and works its way back to the first.

**Errors:** If no item exists, -1 is returned.

See also: IndexOfName (638), TNamedIniCollectionItem.UserData (640)

### 22.7.5 TNamedIniCollection.IndexOfName

**Synopsis:** Search for an item, based on its name, and return its position

**Declaration:**

```pascal
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 its way back to the first.

**Errors:** If no item exists, -1 is returned.

See also: IndexOfUserData (638), TNamedIniCollectionItem.Name (640)
22.7.6 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 (638) 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 (638), FindByUserData (639)

22.7.7 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 (640) property value matches the UserData parameter. If no item is found, Nil is returned.

Errors: If no item is found, Nil is returned.

22.7.8 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 (639) items. The index is zero based.

See also: TNamedIniCollectionItem (639)

22.8 TNamedIniCollectionItem

22.8.1 Description

TNamedIniCollectionItem is a TIniCollectionItem (636) 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 (636)

22.8.2 Property overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Properties</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>640</td>
<td>Name</td>
<td>rw</td>
<td>Name of the item</td>
</tr>
<tr>
<td>640</td>
<td>UserData</td>
<td>rw</td>
<td>User-defined data</td>
</tr>
</tbody>
</table>
22.8.3 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.

22.8.4 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 (640)
Chapter 23

Reference for unit ’IniFiles’

23.1 Used units

Table 23.1: Used units by unit ’IniFiles’

<table>
<thead>
<tr>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Classes</td>
<td>??</td>
</tr>
<tr>
<td>Contnrs</td>
<td>201</td>
</tr>
<tr>
<td>sysutils</td>
<td>??</td>
</tr>
</tbody>
</table>

23.2 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 (643) and is implemented in TIniFile (656) and TMemIniFile (665). The API presented by these components is Delphi compatible.

23.3 Constants, types and variables

23.3.1 Types

TIniFileOption = (ifoStripComments, ifoStripInvalid, ifoEscapeLineFeeds,
                   ifoCaseSensitive, ifoStripQuotes,
                   ifoFormatSettingsActive, ifoWriteStringBoolean)
Table 23.2: Enumeration values for type TIniFileOption

<table>
<thead>
<tr>
<th>Value</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>ifoCaseSensitive</td>
<td>Key and section names are case sensitive</td>
</tr>
<tr>
<td>ifoEscapeLineFeeds</td>
<td>Observe backslash as linefeed escape character</td>
</tr>
<tr>
<td>ifoFormatSettingsActive</td>
<td>Observe the values in FormatSettings</td>
</tr>
<tr>
<td>ifoStripComments</td>
<td>Strip comments from file</td>
</tr>
<tr>
<td>ifoStripInvalid</td>
<td>Strip invalid lines from file</td>
</tr>
<tr>
<td>ifoStripQuotes</td>
<td>Strip double quotes from values</td>
</tr>
<tr>
<td>ifoWriteStringBoolean</td>
<td>Read/Write booleans as strings instead of 0/1</td>
</tr>
</tbody>
</table>

TIniFileOption enumerates the possible options when creating a new TCustomIniFile 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. It is used in the TCustomIniFile.Create constructor and TCustomIniFile.Options property.

TSectionValuesOption = (svoIncludeComments, svoIncludeInvalid, svoIncludeQuotes)

Table 23.3: Enumeration values for type TSectionValuesOption

<table>
<thead>
<tr>
<th>Value</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>svoIncludeComments</td>
<td>Include comment lines</td>
</tr>
<tr>
<td>svoIncludeInvalid</td>
<td>Include invalid lines</td>
</tr>
<tr>
<td>svoIncludeQuotes</td>
<td>Include existing quotes around values</td>
</tr>
</tbody>
</table>

TSectionValuesOption is used to control the behaviour of TCustomIniFile.ReadSectionValues. 

svoIncludeComments Include comment lines
svoIncludeInvalid Include invalid lines
svoIncludeQuotes Include existing quotes around values

TSectionValuesOptions = Set of TSectionValuesOption

TSectionValuesOptions is the set for TSectionValuesOptions. It is used in the TCustomIniFile.ReadSectionValues call.
23.4 TCustomIniFile

23.4.1 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 (656) or TMemIniFile (665).

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 (656), TMemIniFile (665)

23.4.2 Method overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>644</td>
<td>Create</td>
<td>Instantiate a new instance of TCustomIniFile.</td>
</tr>
<tr>
<td>652</td>
<td>DeleteKey</td>
<td>Delete a key from a section</td>
</tr>
<tr>
<td>645</td>
<td>Destroy</td>
<td>Remove the TCustomIniFile instance from memory</td>
</tr>
<tr>
<td>651</td>
<td>EraseSection</td>
<td>Clear a section</td>
</tr>
<tr>
<td>649</td>
<td>ReadBinaryStream</td>
<td>Read binary data</td>
</tr>
<tr>
<td>647</td>
<td>ReadBool</td>
<td></td>
</tr>
<tr>
<td>648</td>
<td>ReadDate</td>
<td>Read a date value</td>
</tr>
<tr>
<td>648</td>
<td>ReadDateTime</td>
<td>Read a Date/Time value</td>
</tr>
<tr>
<td>648</td>
<td>ReadFloat</td>
<td>Read a floating point value</td>
</tr>
<tr>
<td>646</td>
<td>ReadInt64</td>
<td>Read an Int64 value</td>
</tr>
<tr>
<td>646</td>
<td>ReadInteger</td>
<td>Read an integer value from the file</td>
</tr>
<tr>
<td>650</td>
<td>ReadSection</td>
<td>Read the key names in a section</td>
</tr>
<tr>
<td>651</td>
<td>ReadSections</td>
<td>Read the list of sections</td>
</tr>
<tr>
<td>651</td>
<td>ReadSectionValues</td>
<td>Read names and values of a section</td>
</tr>
<tr>
<td>645</td>
<td>ReadString</td>
<td>Read a string valued key</td>
</tr>
<tr>
<td>648</td>
<td>ReadTime</td>
<td>Read a time value</td>
</tr>
<tr>
<td>645</td>
<td>SectionExists</td>
<td>Check if a section exists</td>
</tr>
<tr>
<td>645</td>
<td>SetBoolStringValues</td>
<td>Set the boolean string values to use when writing to file</td>
</tr>
<tr>
<td>652</td>
<td>UpdateFile</td>
<td>Update the file on disk</td>
</tr>
<tr>
<td>652</td>
<td>ValueExists</td>
<td>Check if a value exists</td>
</tr>
<tr>
<td>650</td>
<td>WriteBinaryStream</td>
<td>Write binary data</td>
</tr>
<tr>
<td>647</td>
<td>WriteBool</td>
<td>Write boolean value</td>
</tr>
<tr>
<td>649</td>
<td>WriteDate</td>
<td>Write date value</td>
</tr>
<tr>
<td>649</td>
<td>WriteDateTime</td>
<td>Write date/time value</td>
</tr>
<tr>
<td>650</td>
<td>WriteFloat</td>
<td>Write a floating-point value</td>
</tr>
<tr>
<td>647</td>
<td>WriteInt64</td>
<td>Write an Int64 value</td>
</tr>
<tr>
<td>646</td>
<td>WriteInteger</td>
<td>Write an integer value</td>
</tr>
<tr>
<td>646</td>
<td>WriteString</td>
<td>Write a string value</td>
</tr>
<tr>
<td>650</td>
<td>WriteTime</td>
<td>Write time value</td>
</tr>
</tbody>
</table>
23.4.3 Property overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Properties</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>655</td>
<td>BoolFalseStrings</td>
<td>rw</td>
<td>Strings to recognize as boolean False values.</td>
</tr>
<tr>
<td>654</td>
<td>BoolTrueStrings</td>
<td>rw</td>
<td>Strings to recognize as boolean True values.</td>
</tr>
<tr>
<td>653</td>
<td>CaseSensitive</td>
<td>rw</td>
<td>Are key and section names case sensitive.</td>
</tr>
<tr>
<td>652</td>
<td>Encoding</td>
<td>rw</td>
<td>Encoding of the ini file.</td>
</tr>
<tr>
<td>653</td>
<td>EscapeLineFeeds</td>
<td>r</td>
<td>Should linefeeds be escaped ?</td>
</tr>
<tr>
<td>653</td>
<td>FileName</td>
<td>r</td>
<td>Name of the .ini file</td>
</tr>
<tr>
<td>654</td>
<td>FormatSettingsActive</td>
<td>rw</td>
<td>Is FormatSettings used or not</td>
</tr>
<tr>
<td>653</td>
<td>Options</td>
<td>rw</td>
<td>Options currently in effect</td>
</tr>
<tr>
<td>655</td>
<td>OwnsEncoding</td>
<td>r</td>
<td>Does the ini file instance own the encoding ?</td>
</tr>
<tr>
<td>654</td>
<td>StripQuotes</td>
<td>rw</td>
<td>Should quotes be stripped from string values</td>
</tr>
</tbody>
</table>

23.4.4 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 \nline 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 (653) 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 (645), TCustomIniFile.Options (653)
23.4.5 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 (643)

23.4.6 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 (647). 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 i foWriteStringBoolean is not in the ini file options (653).


23.4.7 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 (652)

23.4.8 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 (646), ReadInteger (646), ReadBool (647), ReadDate (648), ReadDateTime (648), ReadTime (648), ReadFloat (648), ReadBinaryStream (649)
23.4.9 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 (645), WriteInteger (646), WriteBool (647), WriteDate (649), WriteDateTime (649), WriteTime (650), WriteFloat (650), WriteBinaryStream (650)

23.4.10 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 (646), ReadString (645), ReadBool (647), ReadDate (648), ReadDateTime (648), ReadTime (648), ReadFloat (648), ReadBinaryStream (649)

23.4.11 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 (646), WriteString (646), WriteBool (647), WriteDate (649), WriteDateTime (649), WriteTime (650), WriteFloat (650), WriteBinaryStream (650)

23.4.12 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 reads at most a 32-bit value.

See also: TCustomIniFile.ReadInteger, TCustomIniFile.WriteInt64

23.4.13 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, TCustomIniFile.ReadInt64

23.4.14 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, ReadInteger, ReadString, ReadDate, ReadDateTime, ReadTime, ReadFloat, ReadBinaryStream

23.4.15 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, WriteInteger, WriteString, WriteDate, WriteDateTime, WriteTime, WriteFloat, WriteBinaryStream
23.4.16  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 (649), ReadInteger (646), ReadBool (647), ReadString (645), ReadDateTime (648), ReadTime (648), ReadFloat (648), ReadBinaryStream (649)

23.4.17  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 (649), ReadInteger (646), ReadBool (647), ReadDate (648), ReadString (645), ReadTime (648), ReadFloat (648), ReadBinaryStream (649)

23.4.18  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 (650), ReadInteger (646), ReadBool (647), ReadDate (648), ReadString (645), ReadTime (648), ReadBinaryStream (649)

23.4.19  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 (650), ReadInteger (646), ReadBool (647), ReadDate (648), ReadDateTime (648), ReadString (645), ReadFloat (648), ReadBinaryStream (649)

23.4.20 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 (650), ReadInteger (646), ReadBool (647), ReadDate (648), ReadDateTime (648), ReadTime (648), ReadString (645)

23.4.21 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 (648), WriteInteger (646), WriteBool (647), WriteString (646), WriteDateTime (649), WriteTime (650), WriteFloat (650), WriteBinaryStream (650)

23.4.22 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 (648), WriteInteger (646), WriteBool (647), WriteDate (649), WriteString (646), WriteTime (650), WriteFloat (650), WriteBinaryStream (650)
23.4.23 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 (648), WriteInteger (646), WriteBool (647), WriteDate (649), WriteDateTime (649), WriteTime (650), WriteString (646), WriteBinaryStream (650)

23.4.24 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 (648), WriteInteger (646), WriteBool (647), WriteDate (649), WriteDateTime (649), WriteString (646), WriteFloat (650), WriteBinaryStream (650)

23.4.25 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 its size can be determined. The data stream is not repositioned, it must be at the correct position.

See also: ReadBinaryStream (649), WriteInteger (646), WriteBool (647), WriteDate (649), WriteDateTime (649), WriteTime (650), WriteFloat (650), WriteString (646)

23.4.26 TCustomIniFile.ReadSection

Synopsis: Read the key names in a section

Declaration: procedure ReadSection(const Section: string; Strings: TStrings);

Visibility: public

Abstract

650
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 (651) method should be used.

See also: ReadSections (651), SectionExists (645), ReadSectionValues (651)

### 23.4.27 TCustomIniFile.ReadSections

**Synopsis:** Read the list of sections

**Declaration:**

```delphi
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 (645), ReadSectionValues (651), ReadSection (650)

### 23.4.28 TCustomIniFile.ReadSectionValues

**Synopsis:** Read names and values of a section

**Declaration:**

```delphi
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 (650) can be used.

See also: SectionExists (645), ReadSections (651), ReadSection (650)

### 23.4.29 TCustomIniFile.EraseSection

**Synopsis:** Clear a section

**Declaration:**

```delphi
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 (645), ReadSections (651), DeleteKey (652)
23.4.30  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 (651)

23.4.31  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 (660)

23.4.32  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 (645)

23.4.33  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 (644), TCustomIniFile.OwnsEncoding (655)
23.4.34 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 (644)

23.4.35 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 (641) for a list of allowed options. The initial value of this property can be specified using the constructor of the class, TCustomIniFile.Create (644). Not all options can be specified after the ini file object was created.

See also: TIniFileOption (641), TIniFileOptions (642), TCustomIniFile.Create (644)

23.4.36 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 (644), as the value of this property must be specified when the TCustomIniFile instance is created.

By default, EscapeLineFeeds is False.

See also: Create (644), CaseSensitive (653)

23.4.37 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 (653)

23.4.38  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).

23.4.39  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.

23.4.40  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 (647) The first string in the list will be used when writing a True boolean value in TCustomIniFile.WriteBool (647).

The string values are ignored ififoWriteStringBoolean is not in the ini file options (653).

23.4.41 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 (647). The first string in the list will be used when writing a False boolean value in TCustomIniFile.WriteBool (647). The string values are ignored if ifoWriteStringBoolean is not in the ini file options (653).


23.4.42 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 (652), TCustomIniFile.Create (644)

23.5 THashedStringList

23.5.1 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 (656) and IndexOfName (656) functions make use of these hash values to quickly locate a value.

See also: IndexOf (656), IndexOfName (656)

23.5.2 Method overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>655</td>
<td>Destroy</td>
<td>Clean up instance</td>
</tr>
<tr>
<td>656</td>
<td>IndexOf</td>
<td>Returns the index of a string in the list of strings</td>
</tr>
<tr>
<td>656</td>
<td>IndexOfName</td>
<td>Return the index of a name in the list of name=value pairs</td>
</tr>
</tbody>
</table>

23.5.3 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.
23.5.4 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 (656)

23.5.5 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 (656)

23.6 TIniFile

23.6.1 Description

TIniFile is an implementation of TCustomIniFile (643) which does the same as TMemIniFile (665), 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 (643), TMemIniFile (665)

23.6.2 Method overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>657</td>
<td>Create</td>
<td>Create a new instance of TIniFile</td>
</tr>
<tr>
<td>659</td>
<td>DeleteKey</td>
<td>Delete key</td>
</tr>
<tr>
<td>657</td>
<td>Destroy</td>
<td>Remove the TIniFile instance from memory</td>
</tr>
<tr>
<td>659</td>
<td>EraseSection</td>
<td></td>
</tr>
<tr>
<td>658</td>
<td>ReadSection</td>
<td>Read the key names in a section</td>
</tr>
<tr>
<td>658</td>
<td>ReadSectionRaw</td>
<td>Read raw section</td>
</tr>
<tr>
<td>658</td>
<td>ReadSections</td>
<td>Read section names</td>
</tr>
<tr>
<td>659</td>
<td>ReadSectionValues</td>
<td></td>
</tr>
<tr>
<td>657</td>
<td>ReadString</td>
<td>Read a string</td>
</tr>
<tr>
<td>659</td>
<td>UpdateFile</td>
<td>Update the file on disk</td>
</tr>
<tr>
<td>658</td>
<td>WriteString</td>
<td>Write string to file</td>
</tr>
</tbody>
</table>
23.6.3 Property overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Properties</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>660</td>
<td>CacheUpdates</td>
<td>rw</td>
<td>Should changes be kept in memory</td>
</tr>
<tr>
<td>660</td>
<td>Stream</td>
<td>r</td>
<td>Stream from which ini file was read</td>
</tr>
<tr>
<td>660</td>
<td>WriteBOM</td>
<td>rw</td>
<td>Indicates if a Byte Order Mark (BOM) is written at the start of</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>the .INI file</td>
</tr>
</tbody>
</table>

23.6.4 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 (653) property can be specified in AOptions.

See also: TCustomIniFile (643), TMemIniFile (665), TCustomIniFile.Options (653)

23.6.5 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 (652), CacheUpdates (660)

23.6.6 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` abstract method by looking at the in-memory copy of the ini file and returning the string found there.

See also: `TCustomIniFile.ReadString`

### 23.6.7 TIniFile.WriteString

**Synopsis:** Write string to file

**Declaration:**
```pascal
procedure WriteString(const Section: string; const Ident: string; const Value: string); Override
```

**Visibility:** public

**Description:** `WriteString` implements the `TCustomIniFile.WriteString` abstract method by writing the string to the in-memory copy of the ini file. If `CacheUpdates` 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.

### 23.6.8 TIniFile.ReadSection

**Synopsis:** Read the key names in a section

**Declaration:**
```pascal
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`.

See also: `TCustomIniFile.ReadSection`, `TIniFile.ReadSectionRaw`

### 23.6.9 TIniFile.ReadSectionRaw

**Synopsis:** Read raw section

**Declaration:**
```pascal
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`, `TCustomIniFile.ReadSection`

### 23.6.10 TIniFile.ReadSections

**Synopsis:** Read section names

**Declaration:**
```pascal
procedure ReadSections(Strings: TStrings); Override
```

**Visibility:** public

**Description:** `ReadSections` is the implementation of `TCustomIniFile.ReadSections`. It operates on the in-memory copy of the ini file, and places all section names in `Strings`.

See also: `TIniFile.ReadSection`, `TCustomIniFile.ReadSections`, `TIniFile.ReadSectionValues`
23.6.11 TIniFile.ReadSectionValues

Synopsis:

Declaration: procedure ReadSectionValues(const Section: string; Strings: TStrings; AOptions: TSectionValuesOptions); Override

Visibility: public

Description: ReadSectionValues is the implementation of TCustomIniFile.ReadSectionValues (651). 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 (658), TCustomIniFile.ReadSectionValues (651), TIniFile.ReadSections (658)

23.6.12 TIniFile.EraseSection

Synopsis:

Declaration: procedure EraseSection(const Section: string); Override

Visibility: public

Description: EraseSection deletes the section Section from memory, if CacheUpdates (660) is False, then the file is immediately updated on disk. This method is the implementation of the abstract TCustomIniFile.EraseSection (651) method.

See also: TCustomIniFile.EraseSection (651), TIniFile.ReadSection (658), TIniFile.ReadSections (658)

23.6.13 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 (660) is False, then the file is immediately updated on disk.

See also: CacheUpdates (660)

23.6.14 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 (660)
23.6.15  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 method will use this stream to write changes to.

See also: Create, UpdateFile

23.6.16  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. 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

23.6.17  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 UpdateFile method, and is assigned to the WriteBOM property in the TStrings instance used to write the content for the .INI file.
23.7    TIniFileKey

23.7.1 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 (656) class.

See also: TIniFile (656)

23.7.2 Method overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>661</td>
<td>Create</td>
<td>Create a new instance of TIniFileKey</td>
</tr>
</tbody>
</table>

23.7.3 Property overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Properties</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>661</td>
<td>Ident</td>
<td>rw</td>
<td>Key name</td>
</tr>
<tr>
<td>661</td>
<td>Value</td>
<td>rw</td>
<td>Key value</td>
</tr>
</tbody>
</table>

23.7.4 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 (661) with AIdent and Value (661) with AValue.

See also: Ident (661), Value (661)

23.7.5 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 (661)

23.7.6 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 (661)

## 23.8 TIniFileKeyList

### 23.8.1 Description

*TIniFileKeyList* maintains a list of *TIniFileKey* instances on behalf of the *TIniFileSection* class. It stores the keys of one section of the .ini files.

See also: *TIniFileKey* (661), *TIniFileSection* (663)

### 23.8.2 Method overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>662</td>
<td>Clear</td>
<td>Clear the list</td>
</tr>
<tr>
<td>662</td>
<td>Destroy</td>
<td>Free the instance</td>
</tr>
</tbody>
</table>

### 23.8.3 Property overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Properties</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>662</td>
<td>Items</td>
<td>r</td>
<td>Indexed access to <em>TIniFileKey</em> items in the list</td>
</tr>
</tbody>
</table>

### 23.8.4 TIniFileKeyList.Destroy

**Synopsis:** Free the instance

**Declaration:**

```
destructor Destroy; Override
```

**Visibility:** public

**Description:** *Destroy* clears up the list using *Clear* (662) and then calls the inherited destroy.

See also: *Clear* (662)

### 23.8.5 TIniFileKeyList.Clear

**Synopsis:** Clear the list

**Declaration:**

```
procedure Clear; Override
```

**Visibility:** public

**Description:** *Clear* removes all *TIniFileKey* instances from the list, and frees the instances.

See also: *TIniFileKey* (661)

### 23.8.6 TIniFileKeyList.Items

**Synopsis:** Indexed access to *TIniFileKey* items in the list

**Declaration:**

```
Property Items[Index: Integer]: TIniFileKey; default
```
CHAPTER 23. REFERENCE FOR UNIT 'INIFILES'

Visibility: public

Access: Read

Description: Items provides indexed access to the TIniFileKey (661) items in the list. The index is zero-based and runs from 0 to Count-1.

See also: TIniFileKey (661)

23.9 TIniFileSection

23.9.1 Description

TIniFileSection is a class which represents a section in the .ini, and is used internally by the TIniFile (656) class (one instance of TIniFileSection is created for each section in the file by the TIniFileSectionList (664) list). The name of the section is stored in the Name (664) property, and the key/value pairs in this section are available in the KeyList (664) property.

See also: TIniFileKeyList (662), TIniFile (656), TIniFileSectionList (664)

23.9.2 Method overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>663</td>
<td>Create</td>
<td>Create a new section object</td>
</tr>
<tr>
<td>664</td>
<td>Destroy</td>
<td>Free the section object from memory</td>
</tr>
<tr>
<td>663</td>
<td>Empty</td>
<td>Is the section empty</td>
</tr>
</tbody>
</table>

23.9.3 Property overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Properties</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>664</td>
<td>KeyList</td>
<td>r</td>
<td>List of key/value pairs in this section</td>
</tr>
<tr>
<td>664</td>
<td>Name</td>
<td>r</td>
<td>Name of the section</td>
</tr>
</tbody>
</table>

23.9.4 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.

23.9.5 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 (662) instance to keep all the key/value pairs for this section.

See also: TIniFileKeyList (662)
23.9.6 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 (663), TIniFileKeyList (662)

23.9.7 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 (664)

23.9.8 TIniFileSection.KeyList

Synopsis: List of key/value pairs in this section

Declaration: Property KeyList : TIniFileKeyList

Visibility: public

Access: Read

Description: KeyList is the TIniFileKeyList (662) instance that is used by the TIniFileSection to keep the key/value pairs of the section.

See also: TIniFileSection.Name (664), TIniFileKeyList (662)

23.10 TIniFileSectionList

23.10.1 Description

TIniFileSectionList maintains a list of TIniFileSection (663) instances, one for each section in an .ini file. TIniFileSectionList is used internally by the TIniFile (656) class to represent the sections in the file.

See also: TIniFileSection (663), TIniFile (656)

23.10.2 Method overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>665</td>
<td>Clear</td>
<td>Clear the list</td>
</tr>
<tr>
<td>665</td>
<td>Destroy</td>
<td>Free the object from memory</td>
</tr>
</tbody>
</table>
23.10.3 Property overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Properties</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>665</td>
<td>Items</td>
<td>r</td>
<td>Indexed access to all the section objects in the list</td>
</tr>
</tbody>
</table>

23.10.4 TIniFileSectionList.Destroy

Synopsis: Free the object from memory

Declaration: destructor Destroy; Override

Visibility: public

Description: Destroy calls Clear (665) to clear the section list and the calls the inherited Destroy

See also: Clear (665)

23.10.5 TIniFileSectionList.Clear

Synopsis: Clear the list

Declaration: procedure Clear; Override

Visibility: public

Description: Clear removes all TIniFileSection (663) items from the list, and frees the items it removes from the list.

See also: TIniFileSection (663), TIniFileSectionList.Items (665)

23.10.6 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 (663), TIniFileSectionList.Clear (665)

23.11 TMemIniFile

23.11.1 Description

TMemIniFile is a simple descendent of TIniFile (656) 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 (656), TCustomIniFile (643), CacheUpdates (660)
23.11.2 Method overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>666</td>
<td>Clear</td>
<td>Clear the data</td>
</tr>
<tr>
<td>666</td>
<td>Create</td>
<td>Create a new instance of TMemIniFile</td>
</tr>
<tr>
<td>666</td>
<td>GetStrings</td>
<td>Get contents of ini file as stringlist</td>
</tr>
<tr>
<td>667</td>
<td>Rename</td>
<td>Rename the ini file</td>
</tr>
<tr>
<td>667</td>
<td>SetStrings</td>
<td>Set data from a stringlist</td>
</tr>
</tbody>
</table>

23.11.3 TMemIniFile.Create

Synopsis: Create a new instance of TMemIniFile

Declaration:

```delphi
class TMemIniFile
begin
  constructor Create(const AFileName: string; AOptions: TIniFileOptions);
  constructor Create(const AFileName: string; AEscapeLineFeeds: Boolean);
end;
```

Visibility: public

Description: Create simply calls the inherited Create (657), and sets the CacheUpdates (660) to True so updates will be kept in memory till they are explicitly written to disk.

See also: TIniFile.Create (657), CacheUpdates (660)

23.11.4 TMemIniFile.Clear

Synopsis: Clear the data

Declaration: procedure Clear

Visibility: public

Description: Clear removes all sections and key/value pairs from memory. If CacheUpdates (660) is set to False then the file on disk will immediately be emptied.

See also: SetStrings (667), GetStrings (666)

23.11.5 TMemIniFile.GetString

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 (667), which can be used to copy data between 2 in-memory ini files.

See also: SetStrings (667), Clear (666)
23.11.6  **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 (659)`

23.11.7  **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 (666)`. The two calls combined can be used to copy data between 2 in-memory ini files.

See also: `GetStrings (666), Clear (666)`

23.12  **TStringHash**

23.12.1  **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 (656), TStringHash.Create (668), TStringHash.ValueOf (669)`

23.12.2  **Method overview**

<table>
<thead>
<tr>
<th>Page</th>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>668</td>
<td>Add</td>
<td>Add a new value to the hash</td>
</tr>
<tr>
<td>668</td>
<td>Clear</td>
<td>Remove all values</td>
</tr>
<tr>
<td>668</td>
<td>Create</td>
<td>Create a new instance of <code>TStringHash</code></td>
</tr>
<tr>
<td>668</td>
<td>Destroy</td>
<td>Free <code>TStringHash</code> instance</td>
</tr>
<tr>
<td>669</td>
<td>Modify</td>
<td>Try to modify an existing value</td>
</tr>
<tr>
<td>669</td>
<td>Remove</td>
<td>Remove a key from the hash</td>
</tr>
<tr>
<td>669</td>
<td>ValueOf</td>
<td>Retrieve value of <code>Key</code></td>
</tr>
</tbody>
</table>

23.12.3  **Property overview**

<table>
<thead>
<tr>
<th>Page</th>
<th>Properties</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>669</td>
<td><code>AddReplacesExisting</code></td>
<td>rw</td>
<td>Should <code>Add</code> replace existing values or not</td>
</tr>
</tbody>
</table>
23.12.4 **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** (669), **TStringHash.Destroy** (668)

23.12.5 **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** (668), **TStringHash.Clear** (668)

23.12.6 **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** (669). 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** (669), **TStringHash.Modify** (669), **TStringHash.Remove** (669), **TStringHash.ValueOf** (669)

23.12.7 **TStringHash.Clear**

**Synopsis:** Remove all values

**Declaration:**
```
procedure Clear
```

**Visibility:** public

**Description:** Clear removes all values from the hash.

See also: **TStringHash.Destroy** (668), **TStringHash.Add** (668), **TStringHash.Remove** (669), **TStringHash.ValueOf** (669)
23.12.8 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 (668), TStringHash.Clear (668), TStringHash.Remove (669), TStringHash.ValueOf (669)

23.12.9 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 (668), TStringHash.Clear (668), TStringHash.Modify (669), TStringHash.ValueOf (669)

23.12.10 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 (668), TStringHash.Clear (668), TStringHash.Modify (669), TStringHash.Remove (669)

23.12.11 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 (668) will replace an existing value (True) or will raise an exception when an existing value is added (False).

See also: TStringHash.Add (668)
Chapter 24

Reference for unit ’iostream’

24.1 Used units

Table 24.1: Used units by unit ’iostream’

<table>
<thead>
<tr>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Classes</td>
<td>??</td>
</tr>
<tr>
<td>System</td>
<td>??</td>
</tr>
</tbody>
</table>

24.2 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`).

24.3 Constants, types and variables

24.3.1 Types

`TIOSType = (iosInput, iosOutPut, iosError)`

<table>
<thead>
<tr>
<th>Value</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>iosError</td>
<td>The stream can be used to write to standard diagnostic output</td>
</tr>
<tr>
<td>iosInput</td>
<td>The stream can be used to read from standard input</td>
</tr>
<tr>
<td>iosOutPut</td>
<td>The stream can be used to write to standard output</td>
</tr>
</tbody>
</table>

`TIOSType` is passed to the Create (671) constructor of TIOStream (671), it determines what kind of stream is created.
24.4 EIOStreamError

24.4.1 Description
Error thrown in case of an invalid operation on a TIOStream (671).

24.5 TIOStream

24.5.1 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 (670) 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 (672) behaviour based on this position.

See also: TIOSType (670), THandleStream (??)

24.5.2 Method overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>671</td>
<td>Create</td>
<td>Construct a new instance of TIOStream (671)</td>
</tr>
<tr>
<td>671</td>
<td>Read</td>
<td>Read data from the stream.</td>
</tr>
<tr>
<td>672</td>
<td>Seek</td>
<td>Set the stream position</td>
</tr>
<tr>
<td>672</td>
<td>Write</td>
<td>Write data to the stream</td>
</tr>
</tbody>
</table>

24.5.3 TIOStream.Create

Synopsis: Construct a new instance of TIOStream (671)

Declaration: constructor Create(aIOSType: TIOSType)

Visibility: public

Description: Create creates a new instance of TIOStream (671), 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 (671), TIOStream.Write (672)

24.5.4 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 (671) 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 (670), TIOStream.Write (672)

24.5.5 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 (671) 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 (670), TIOStream.Read (671)

24.5.6 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 (671) exception is raised if the stream does not allow the requested seek operation.

See also: EIOStreamError (671)
Chapter 25

Reference for unit ’libtar’

25.1 Used units

Table 25.1: Used units by unit ’libtar’

<table>
<thead>
<tr>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>BaseUnix</td>
<td>??</td>
</tr>
<tr>
<td>Classes</td>
<td>??</td>
</tr>
<tr>
<td>System</td>
<td>??</td>
</tr>
<tr>
<td>sysutils</td>
<td>??</td>
</tr>
<tr>
<td>Unix</td>
<td>??</td>
</tr>
<tr>
<td>UnixType</td>
<td>??</td>
</tr>
</tbody>
</table>

25.2 Overview

The libtar units provides 2 classes to read and write .tar archives: TTarArchive (677) class can be used to read a tar file, and the TTarWriter (679) class can be used to write a tar file. The unit was implemented originally by Stefan Heymann.

25.3 Constants, types and variables

25.3.1 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]

EXECUTE_PERMISSIONS is a set constant containing all possible execute permissions set for an archive entry.
CHAPTER 25. REFERENCE FOR UNIT 'LIBTAR'


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.

25.3.2 Types

TFileType = (ftNormal, ftLink, ftSymbolicLink, ftCharacter, ftBlock, ftDirectory, ftFifo, ftContiguous, ftDumpDir, ftMultiVolume, ftVolumeHeader)

Table 25.2: Enumeration values for type TFileType

<table>
<thead>
<tr>
<th>Value</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>ftBlock</td>
<td>Block device file</td>
</tr>
<tr>
<td>ftCharacter</td>
<td>Character device file</td>
</tr>
<tr>
<td>ftContiguous</td>
<td>Contiguous file</td>
</tr>
<tr>
<td>ftDirectory</td>
<td>Directory</td>
</tr>
<tr>
<td>ftDumpDir</td>
<td>List of files</td>
</tr>
<tr>
<td>ftFifo</td>
<td>FIFO file</td>
</tr>
<tr>
<td>ftLink</td>
<td>Hard link</td>
</tr>
<tr>
<td>ftMultiVolume</td>
<td>Multi-volume file part</td>
</tr>
<tr>
<td>ftNormal</td>
<td>Normal file</td>
</tr>
<tr>
<td>ftSymbolicLink</td>
<td>Symbolic link</td>
</tr>
<tr>
<td>ftVolumeHeader</td>
<td>Volume header, can appear only as first entry in the archive</td>
</tr>
</tbody>
</table>

TFileType describes the file type of a file in the archive. It is used in the FileType field of the TTarDirRec (675) record.

TTarDirRec = record
public
  Name : AnsiString;
  Size : Int64;
  DateTime : TDateTime;
  Permissions : TTarPermissions;
  FileType : TFileType;
CHAPTER 25. REFERENCE FOR UNIT 'LIBTAR'

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 (678) call.

TTarMode = {tmSetUid, tmSetGid, tmSaveText}

Table 25.3: Enumeration values for type TTarMode

<table>
<thead>
<tr>
<th>Value</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>tmSaveText</td>
<td>Bit $200 is set</td>
</tr>
<tr>
<td>tmSetGid</td>
<td>File has SetGID bit set</td>
</tr>
<tr>
<td>tmSetUid</td>
<td>File has SetUID bit set</td>
</tr>
</tbody>
</table>

TTarMode describes extra file modes. It is used in the Mode field of the TTarDirRec (675) record.

TTarModes = Set of TTarMode

TTarModes denotes the full set of permission bits for the file in the field Mode field of the TTarDirRec (675) record.

TTarPermission = (tpReadByOwner, tpWriteByOwner, tpExecuteByOwner,
                  tpReadByGroup, tpWriteByGroup, tpExecuteByGroup,
                  tpReadByOther, tpWriteByOther, tpExecuteByOther)

Table 25.4: Enumeration values for type TTarPermission

<table>
<thead>
<tr>
<th>Value</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>tpExecuteByGroup</td>
<td>Group can execute the file</td>
</tr>
<tr>
<td>tpExecuteByOther</td>
<td>Other people can execute the file</td>
</tr>
<tr>
<td>tpExecuteByOwner</td>
<td>Owner can execute the file</td>
</tr>
<tr>
<td>tpReadByGroup</td>
<td>Group can read the file</td>
</tr>
<tr>
<td>tpReadByOther</td>
<td>Other people can read the file.</td>
</tr>
<tr>
<td>tpReadByOwner</td>
<td>Owner can read the file</td>
</tr>
<tr>
<td>tpWriteByGroup</td>
<td>Group can write the file</td>
</tr>
<tr>
<td>tpWriteByOther</td>
<td>Other people can write the file</td>
</tr>
<tr>
<td>tpWriteByOwner</td>
<td>Owner can write the file</td>
</tr>
</tbody>
</table>
TTarPermission denotes part of a file's 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 (675) record.

### 25.4 Procedures and functions

#### 25.4.1 ClearDirRec

**Synopsis:** Initialize tar archive entry

**Declaration:**

```pascal
procedure ClearDirRec(var DirRec: TTarDirRec)
```

**Visibility:** default

**Description:** ClearDirRec clears the DirRec entry, it basically zeroes out all fields.

See also: TTarDirRec (675)

#### 25.4.2 ConvertFilename

**Synopsis:** Convert filename to archive format

**Declaration:**

```pascal
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.

#### 25.4.3 FileTimeGMT

**Synopsis:** Extract filetime

**Declaration:**

```pascal
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 (675)

#### 25.4.4 PermissionString

**Synopsis:** Convert a set of permissions to a string

**Declaration:**

```pascal
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 (676)

676
25.5  TTarArchive

25.5.1  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 (679), FindNext (678)

25.5.2  Method overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>677</td>
<td>Create</td>
<td>Create a new instance of the archive</td>
</tr>
<tr>
<td>677</td>
<td>Destroy</td>
<td>Destroy TTarArchive instance</td>
</tr>
<tr>
<td>678</td>
<td>FindNext</td>
<td>Find next archive entry</td>
</tr>
<tr>
<td>678</td>
<td>GetFilePos</td>
<td>Return current archive position</td>
</tr>
<tr>
<td>678</td>
<td>ReadFile</td>
<td>Read a file from the archive</td>
</tr>
<tr>
<td>677</td>
<td>Reset</td>
<td>Reset archive</td>
</tr>
<tr>
<td>679</td>
<td>SetFilePos</td>
<td>Set position in archive</td>
</tr>
</tbody>
</table>

25.5.3  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 (678)

25.5.4  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 in-
stance.

See also: TTarArchive.Create (677)

25.5.5  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 (677)

### 25.5.6 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 (678)

### 25.5.7 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.

### 25.5.8 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 (679), TTarArchive.Reset (677)
25.5.9 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 (677), TTarArchive.GetFilePos (678)

25.6 TTarWriter

25.6.1 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 (677)

25.6.2 Method overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>681</td>
<td>AddDir</td>
<td>Add directory to archive</td>
</tr>
<tr>
<td>680</td>
<td>AddFile</td>
<td>Add a file to the archive</td>
</tr>
<tr>
<td>682</td>
<td>AddLink</td>
<td>Add hard link to archive</td>
</tr>
<tr>
<td>680</td>
<td>AddStream</td>
<td>Add stream contents to archive,</td>
</tr>
<tr>
<td>681</td>
<td>AddString</td>
<td>Add string as file data</td>
</tr>
<tr>
<td>681</td>
<td>AddSymbolicLink</td>
<td>Add a symbolic link to the archive</td>
</tr>
<tr>
<td>682</td>
<td>AddVolumeHeader</td>
<td>Add volume header entry</td>
</tr>
<tr>
<td>679</td>
<td>Create</td>
<td>Create a new archive</td>
</tr>
<tr>
<td>680</td>
<td>Destroy</td>
<td>Close archive and clean up TTarWriter</td>
</tr>
<tr>
<td>682</td>
<td>Finalize</td>
<td>Finalize the archive</td>
</tr>
</tbody>
</table>

25.6.3 Property overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Properties</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>683</td>
<td>GID</td>
<td>rw</td>
<td>Archive entry group ID</td>
</tr>
<tr>
<td>683</td>
<td>GroupName</td>
<td>rw</td>
<td>Archive entry group name</td>
</tr>
<tr>
<td>684</td>
<td>Magic</td>
<td>rw</td>
<td>Archive entry Magic constant</td>
</tr>
<tr>
<td>684</td>
<td>Mode</td>
<td>rw</td>
<td>Archive entry mode</td>
</tr>
<tr>
<td>682</td>
<td>Permissions</td>
<td>rw</td>
<td>Archive entry permissions</td>
</tr>
<tr>
<td>683</td>
<td>UID</td>
<td>rw</td>
<td>Archive entry user ID</td>
</tr>
<tr>
<td>683</td>
<td>UserName</td>
<td>rw</td>
<td>Archive entry user name</td>
</tr>
</tbody>
</table>

25.6.4 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` (680)

### 25.6.5 `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` (682)

### 25.6.6 `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.


### 25.6.7 `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.

CHAPTER 25. REFERENCE FOR UNIT 'LIBTAR'

25.6.8 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.


25.6.9 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.


25.6.10 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.

25.6.11 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.


25.6.12 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.


25.6.13 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 (680)

25.6.14 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 (675)
25.6.15   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 (675)

25.6.16   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 (675)

25.6.17   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 (675)

25.6.18   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 (675)
25.6.19  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 (675)

25.6.20  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 (675)
Chapter 26

Reference for unit ’memds’

26.1 Used units

Table 26.1: Used units by unit ’memds’

<table>
<thead>
<tr>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Classes</td>
<td>??</td>
</tr>
<tr>
<td>DB</td>
<td>319</td>
</tr>
<tr>
<td>System</td>
<td>??</td>
</tr>
<tr>
<td>sysutils</td>
<td>??</td>
</tr>
<tr>
<td>Types</td>
<td>??</td>
</tr>
</tbody>
</table>

26.2 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)

26.3 Constants, types and variables

26.3.1 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.

### 26.4 MDSError

#### 26.4.1 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
26.5 TMemDataset

26.5.1 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
- ft WideString
- 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 (375)

### 26.5.2 Method overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>690</td>
<td>BookmarkValid</td>
<td>Determines if the specified Bookmark is valid</td>
</tr>
<tr>
<td>692</td>
<td>Clear</td>
<td>Clears the content in the in-memory dataset</td>
</tr>
<tr>
<td>690</td>
<td>CompareBookmarks</td>
<td>Gets the relative order for the specified Bookmarks</td>
</tr>
<tr>
<td>694</td>
<td>CopyFromDataset</td>
<td>Loads field definitions and optional data from the specified TDataset</td>
</tr>
<tr>
<td>689</td>
<td>Create</td>
<td>Constructor for the class instance</td>
</tr>
<tr>
<td>690</td>
<td>CreateBlobStream</td>
<td>Creates a stream used to read or write Blob field data in the in-memory dataset</td>
</tr>
<tr>
<td>691</td>
<td>CreateTable</td>
<td>Creates the internal storage for records in the in-memory dataset</td>
</tr>
<tr>
<td>692</td>
<td>DataSize</td>
<td>Size of the internal TMemoryStream used in the in-memory dataset</td>
</tr>
<tr>
<td>689</td>
<td>Destroy</td>
<td>Destructor for the class instance</td>
</tr>
<tr>
<td>693</td>
<td>LoadFromFile</td>
<td>Loads the content for the dataset from the specified file name</td>
</tr>
<tr>
<td>693</td>
<td>LoadFromStream</td>
<td>Loads the content for the dataset from the specified stream</td>
</tr>
<tr>
<td>691</td>
<td>Locate</td>
<td>Locates a record with the specified values in the in-memory dataset</td>
</tr>
<tr>
<td>691</td>
<td>Lookup</td>
<td>Searches for a record with the specified values, and returns a list of values</td>
</tr>
<tr>
<td>692</td>
<td>SaveToFile</td>
<td>Saves field definitions and optional record data to the specified file name</td>
</tr>
<tr>
<td>693</td>
<td>SaveToStream</td>
<td>Saves field definitions and optional record data to the specified stream</td>
</tr>
</tbody>
</table>
### 26.5.3 Property overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Properties</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>696</td>
<td>Active</td>
<td></td>
<td>Indicates if the in-memory dataset is Active</td>
</tr>
<tr>
<td>698</td>
<td>AfterCancel</td>
<td></td>
<td></td>
</tr>
<tr>
<td>697</td>
<td>AfterClose</td>
<td></td>
<td></td>
</tr>
<tr>
<td>698</td>
<td>AfterDelete</td>
<td></td>
<td></td>
</tr>
<tr>
<td>697</td>
<td>AfterEdit</td>
<td></td>
<td></td>
</tr>
<tr>
<td>697</td>
<td>AfterInsert</td>
<td></td>
<td></td>
</tr>
<tr>
<td>696</td>
<td>AfterOpen</td>
<td></td>
<td></td>
</tr>
<tr>
<td>698</td>
<td>AfterPost</td>
<td></td>
<td></td>
</tr>
<tr>
<td>699</td>
<td>AfterScroll</td>
<td></td>
<td></td>
</tr>
<tr>
<td>698</td>
<td>BeforeCancel</td>
<td></td>
<td></td>
</tr>
<tr>
<td>697</td>
<td>BeforeClose</td>
<td></td>
<td></td>
</tr>
<tr>
<td>698</td>
<td>BeforeDelete</td>
<td></td>
<td></td>
</tr>
<tr>
<td>697</td>
<td>BeforeEdit</td>
<td></td>
<td></td>
</tr>
<tr>
<td>697</td>
<td>BeforeInsert</td>
<td></td>
<td></td>
</tr>
<tr>
<td>696</td>
<td>BeforeOpen</td>
<td></td>
<td></td>
</tr>
<tr>
<td>698</td>
<td>BeforePost</td>
<td></td>
<td></td>
</tr>
<tr>
<td>699</td>
<td>BeforeScroll</td>
<td></td>
<td></td>
</tr>
<tr>
<td>696</td>
<td>FieldDefs</td>
<td></td>
<td>Field definitions for the in-memory dataset</td>
</tr>
<tr>
<td>695</td>
<td>FileModified</td>
<td>r</td>
<td>Indicates if the in-memory dataset has been modified</td>
</tr>
<tr>
<td>695</td>
<td>FileName</td>
<td>rw</td>
<td>File name used to read or write field definitions and optional data</td>
</tr>
<tr>
<td>695</td>
<td>Filter</td>
<td></td>
<td>Filter for the dataset</td>
</tr>
<tr>
<td>696</td>
<td>Filtered</td>
<td></td>
<td>Indicates if records in the dataset are filtered using OnFilter-Record</td>
</tr>
<tr>
<td>689</td>
<td>OnDeleteError</td>
<td></td>
<td></td>
</tr>
<tr>
<td>699</td>
<td>OnEditError</td>
<td></td>
<td></td>
</tr>
<tr>
<td>700</td>
<td>OnFilterRecord</td>
<td></td>
<td>Event handler signalled to include or exclude records in the in-memory dataset</td>
</tr>
<tr>
<td>699</td>
<td>OnNewRecord</td>
<td></td>
<td></td>
</tr>
<tr>
<td>700</td>
<td>OnPostError</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### 26.5.4 TMemDataset.Create

**Synopsis:** Constructor for the class instance

**Declaration:**

```pascal
constructor Create(AOwner: TComponent); Override
```

**Visibility:** public

**Description:** Create is the overridden constructor for the class instance. Create calls the inherited constructor. Create allocates resources required for internal member variables in the class, such as the TMemoryStream that contains the field definitions and record data and the list used to store Blob data. Other internal member variables are set to their default values. Creates sets the default values for the following published properties:

- **BookmarkSize**:
  - SizeOf(LongInt)

### 26.5.5 TMemDataset.Destroy

**Synopsis:** Destructor for the class instance

**Declaration:**

```pascal
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.

26.5.6 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.

26.5.7 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.

26.5.8 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 (454), TDataset.State (401), TField.ReadOnly (448)
26.5.9  **TMemDataset.Locate**

Synopsis: Locates a record with the specified values in the in-memory dataset

Declaration: 
```pascal
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 (390), TLocateOption (330), TLocateOptions (330), TDataset.Resync (393)

26.5.10  **TMemDataset.Lookup**

Synopsis: Searches for a record with the specified values, and returns a list of values

Declaration: 
```pascal
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:
```pascal
var AResultVals: Variant;
AResultVals := AMemDS.Lookup('lastname, firstname', VarArrayCreate('Franks', 'Peter'), 'lastname, firstname, birthdate');
```

See also: TMemDataset.Locate (691)

26.5.11  **TMemDataset.CreateTable**

Synopsis: Creates the internal storage for records in the in-memory dataset

Declaration: 
```
procedure CreateTable
```

Visibility: public
CHAPTER 26. REFERENCE FOR UNIT 'MEMDS'

**CHAPTER 26. REFERENCE FOR UNIT 'MEMDS'**

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 (403), TMemDataset.Clear (692), TDataset.FieldDefs (398), CreateTable (691)

### 26.5.12 TMemDataset.DataSize

**Synopsis:** Size of the internal TMemoryStream used in the in-memory dataset

**Declaration:**

```delphi
definition 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.

### 26.5.13 TMemDataset.Clear

**Synopsis:** Clears the content in the in-memory dataset

**Declaration:**

```delphi
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 (403), TDataset.Close (382), TDataset.FieldDefs (398)

### 26.5.14 TMemDataset.SaveToFile

**Synopsis:** Saves field definitions and optional record data to the specified file name

**Declaration:**

```delphi
procedure SaveToFile(AFileName: string)
procedure SaveToFile(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.
26.5.15 TMemDataset.SaveToStream

**Synopsis:** Saves field definitions and optional record data to the specified stream

**Declaration:**

```pascal
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 (693)

26.5.16 TMemDataset.LoadFromStream

**Synopsis:** Loads the content for the dataset from the specified stream

**Declaration:**

```pascal
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 (693)

26.5.17 TMemDataset.LoadFromFile

**Synopsis:** Loads the content for the dataset from the specified file name

**Declaration:**

```pascal
procedure LoadFromFile(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 (693), TMemDataset.SaveToFile (692)
26.5.18 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 FieldDefs 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 (401), TDataSet.FieldDefs (398), TMemDataset.CreateTable (691), TDataSet.Append (380), TDataSet.Post (392)
26.5.19  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

26.5.20  TMemDataset.Filter

Synopsis: Filter for the dataset

Declaration: Property Filter : ; unimplemented;

Visibility: public

Access:

Remark: Filter is not implemented in TMemDataset. Values assigned to the Filter property are silently discarded. Use OnFilterRecord and Filtered instead.

See also: TMemDataset.Filtered (696), TMemDataset.OnFilterRecord (700)

26.5.21  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.
26.5.22  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 (700), TDataset.Filter (402)

26.5.23  TMemDataset.Active
   Synopsis: Indicates if the in-memory dataset is Active
   Declaration: Property Active :
      Visibility: published
      Access:

26.5.24  TMemDataset.FieldDefs
   Synopsis: Field definitions for the in-memory dataset
   Declaration: Property FieldDefs :
      Visibility: published
      Access:

26.5.25  TMemDataset.BeforeOpen
   Synopsis:
   Declaration: Property BeforeOpen :
      Visibility: published
      Access:

26.5.26  TMemDataset.AfterOpen
   Synopsis:
   Declaration: Property AfterOpen :
      Visibility: published
      Access:
26.5.27 TMemDataset.BeforeClose
Synopsis:
Declaration: Property BeforeClose :
  Visibility: published
  Access:

26.5.28 TMemDataset.AfterClose
Synopsis:
Declaration: Property AfterClose :
  Visibility: published
  Access:

26.5.29 TMemDataset.BeforeInsert
Synopsis:
Declaration: Property BeforeInsert :
  Visibility: published
  Access:

26.5.30 TMemDataset.AfterInsert
Synopsis:
Declaration: Property AfterInsert :
  Visibility: published
  Access:

26.5.31 TMemDataset.BeforeEdit
Synopsis:
Declaration: Property BeforeEdit :
  Visibility: published
  Access:

26.5.32 TMemDataset.AfterEdit
Synopsis:
Declaration: Property AfterEdit :
  Visibility: published
  Access:
26.5.33 TMemDataset.BeforePost
Synopsis:
Declaration: Property BeforePost :
   Visibility: published
   Access:

26.5.34 TMemDataset.AfterPost
Synopsis:
Declaration: Property AfterPost :
   Visibility: published
   Access:

26.5.35 TMemDataset.BeforeCancel
Synopsis:
Declaration: Property BeforeCancel :
   Visibility: published
   Access:

26.5.36 TMemDataset.AfterCancel
Synopsis:
Declaration: Property AfterCancel :
   Visibility: published
   Access:

26.5.37 TMemDataset.BeforeDelete
Synopsis:
Declaration: Property BeforeDelete :
   Visibility: published
   Access:

26.5.38 TMemDataset.AfterDelete
Synopsis:
Declaration: Property AfterDelete :
   Visibility: published
   Access:
26.5.39  TMemDataset.BeforeScroll

Synopsis:

Declaration: Property BeforeScroll :

Visibility: published

Access:

26.5.40  TMemDataset.AfterScroll

Synopsis:

Declaration: Property AfterScroll :

Visibility: published

Access:

26.5.41  TMemDataset.OnDeleteError

Synopsis:

Declaration: Property OnDeleteError :

Visibility: published

Access:

26.5.42  TMemDataset.OnEditError

Synopsis:

Declaration: Property OnEditError :

Visibility: published

Access:

Description:

See also: (??)

26.5.43  TMemDataset.OnNewRecord

Synopsis:

Declaration: Property OnNewRecord :

Visibility: published

Access:
26.5.44 TMemDataset.OnPostError

Synopsis:

Declaration: Property OnPostError :

Visibility: published

Access:

26.5.45 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 (410), TFilterRecordEvent (329), TDataSet.Filtered (402), TDataSet.Resync (393)
Chapter 27

Reference for unit ’MSSQLConn’

27.1 Used units

Table 27.1: Used units by unit ’MSSQLConn’

<table>
<thead>
<tr>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>BufDataset</td>
<td>131</td>
</tr>
<tr>
<td>Classes</td>
<td>??</td>
</tr>
<tr>
<td>DB</td>
<td>319</td>
</tr>
<tr>
<td>dblib</td>
<td>??</td>
</tr>
<tr>
<td>SQLDB</td>
<td>786</td>
</tr>
<tr>
<td>System</td>
<td>??</td>
</tr>
<tr>
<td>sysutils</td>
<td>??</td>
</tr>
</tbody>
</table>

27.2 Overview

Connector to Microsoft SQL Server databases. Needs FreeTDS dblib library.

27.3 Constants, types and variables

27.3.1 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.

27.4 EMSSQLDatabaseError

27.4.1 Description

Sybase/MS SQL Server specific error
27.4.2 Property overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Properties</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>702</td>
<td>DBErrorCode</td>
<td>r</td>
<td>Sybase/MS SQL Server error code</td>
</tr>
</tbody>
</table>

27.4.3 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.

27.5 TMSSQLConnection

27.5.1 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/N-VARCHAR 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')
27.5.2 Method overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>703</td>
<td>Create</td>
<td>Create a new instance of TMSSQLConnection</td>
</tr>
<tr>
<td>703</td>
<td>CreateDB</td>
<td>Create a new MS SQL database</td>
</tr>
<tr>
<td>704</td>
<td>DropDB</td>
<td>Drop a MS SQL database</td>
</tr>
<tr>
<td>703</td>
<td>GetConnectionInfo</td>
<td>Return some information about the connection</td>
</tr>
</tbody>
</table>

27.5.3 Property overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Properties</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>705</td>
<td>CharSet</td>
<td></td>
<td></td>
</tr>
<tr>
<td>705</td>
<td>Connected</td>
<td></td>
<td>Is the connection active</td>
</tr>
<tr>
<td>706</td>
<td>DatabaseName</td>
<td></td>
<td></td>
</tr>
<tr>
<td>705</td>
<td>HostName</td>
<td></td>
<td>Host and optionally port or instance</td>
</tr>
<tr>
<td>706</td>
<td>KeepConnection</td>
<td></td>
<td>Keep connection alive</td>
</tr>
<tr>
<td>706</td>
<td>LoginPrompt</td>
<td></td>
<td>Show login prompt</td>
</tr>
<tr>
<td>707</td>
<td>OnLogin</td>
<td></td>
<td>Called when logging in</td>
</tr>
<tr>
<td>706</td>
<td>Params</td>
<td></td>
<td></td>
</tr>
<tr>
<td>704</td>
<td>Password</td>
<td></td>
<td>Role for user</td>
</tr>
<tr>
<td>704</td>
<td>Role</td>
<td></td>
<td>Role for user</td>
</tr>
<tr>
<td>704</td>
<td>Transaction</td>
<td></td>
<td>Default transaction</td>
</tr>
<tr>
<td>704</td>
<td>UserName</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

27.5.4 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.

27.5.5 TMSSQLConnection.GetConnectionInfo

Synopsis: Return some information about the connection

Declaration: function GetConnectionInfo(InfoType: TConnInfoType) : string; Override

Visibility: public

Description: GetConnectionInfo overrides TSQLConnection.GetConnectionInfo (817) to return the relevant information for the Interbase/Firebird connection.

See also: TSQLConnection.GetConnectionInfo (817), TConnInfoType (796)

27.5.6 TMSSQLConnection.CreateDB

Synopsis: Create a new MS SQL database

Declaration: procedure CreateDB; Override

Visibility: public
CHAPTER 27. REFERENCE FOR UNIT ‘MSSQLCONN’

Description: `CreateDB` creates a database on the server with given `DatabaseName`.

See also: `TMSSQLConnection.DropDB (704)`

### 27.5.7 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 (703)`

### 27.5.8 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).

### 27.5.9 TMSSQLConnection.Transaction

Synopsis: Default transaction

Declaration: `Property Transaction :`

Visibility: published

Access:

Description: `Transaction` is redeclared from `TSQLConnection.Transaction (701)`

See also: `TSQLConnection.Transaction (701)`

### 27.5.10 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).
27.5.11 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.

27.5.12 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.: 127.0.0.1\SQLEXPRESS. Port should be specified with a colon, e.g. BIGBADSERVER:1433

See http://www.freetds.org/userguide/portoverride.htm

27.5.13 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.

27.5.14 TMSSQLConnection.Role

Synopsis: Role for user

Declaration: Property Role :

Visibility: published

Access:

Description: Role is redeclared from TSQLConnection.Role (701)
27.5.15 TMSSQLConnection.DatabaseName

Declaration: Property DatabaseName :

Visibility: published
Access:

Description: TMSSQLConnection specific: the master database should always exist on a server.

27.5.16 TMSSQLConnection.KeepConnection

Synopsis: Keep connection alive

Declaration: Property KeepConnection :

Visibility: published
Access:

Description: KeepConnection is redeclared from TSQLConnection.KeepConnection (701)
See also: TSQLConnection.KeepConnection (701)

27.5.17 TMSSQLConnection.LoginPrompt

Synopsis: Show login prompt

Declaration: Property LoginPrompt :

Visibility: published
Access:

Description: LoginPrompt is redeclared from TSQLConnection.LoginPrompt (701)
See also: TSQLConnection.LoginPrompt (701)

27.5.18 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.
27.5.19 TMSSQLConnection.OnLogin

Synopsis: Called when logging in

Declaration: Property OnLogin:

Visibility: published

Access:

Description: OnLogin is redeclared from TSQLConnection.OnLogin (701)

See also: TSQLConnection.OnLogin (701)

27.6 TMSSQLConnectionDef

27.6.1 Description

Describes the MS SQL connection properties for TSQLConnector (701)

See also: TMSSQLConnection (702), TSQLConnector (701)

27.6.2 Method overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>707</td>
<td>ConnectionClass</td>
<td>Connection class to use.</td>
</tr>
<tr>
<td>708</td>
<td>DefaultLibraryName</td>
<td>Default name of the MSSQL client library</td>
</tr>
<tr>
<td>708</td>
<td>Description</td>
<td>Short description of connection</td>
</tr>
<tr>
<td>708</td>
<td>LoadedLibraryName</td>
<td>Actually loaded library name</td>
</tr>
<tr>
<td>708</td>
<td>LoadFunction</td>
<td>Return Function to call when loading MS-SQL support</td>
</tr>
<tr>
<td>707</td>
<td>TypeName</td>
<td>Connection type name</td>
</tr>
<tr>
<td>708</td>
<td>UnLoadFunction</td>
<td>Return Function to call when unloading MS-SQL support</td>
</tr>
</tbody>
</table>

27.6.3 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.

27.6.4 TMSSQLConnectionDef.ConnectionClass

Synopsis: Connection class to use.

Declaration: class function ConnectionClass: TSQLConnectionClass; Override

Visibility: default

Description: ConnectionClass returns TMSSQLConnection (702)

See also: TMSSQLConnection (702)
### 27.6.5 TMSSQLConnectionDef.Description

**Synopsis:** Short description of connection

**Declaration:**
```pascal
class function Description : string; Override
```

**Visibility:** default

**Description:** Description describes the MS SQL connector type.

### 27.6.6 TMSSQLConnectionDef.DefaultLibraryName

**Synopsis:** Default name of the MSSQL client library

**Declaration:**
```pascal
class function DefaultLibraryName : string; Override
```

**Visibility:** default

**Description:** DefaultLibraryName returns the library name to use when loading the MSSQL client library.

### 27.6.7 TMSSQLConnectionDef.LoadFunction

**Synopsis:** Return Function to call when loading MS-SQL support

**Declaration:**
```pascal
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.

### 27.6.8 TMSSQLConnectionDef.UnLoadFunction

**Synopsis:** Return Function to call when unloading MS-SQL support

**Declaration:**
```pascal
class function UnLoadFunction : TLibraryUnLoadFunction; Override
```

**Visibility:** default

**Description:** UnLoadFunction is used by the connector logic to get the function to unload MS-SQL support.

### 27.6.9 TMSSQLConnectionDef.LoadedLibraryName

**Synopsis:** Actually loaded library name

**Declaration:**
```pascal
class function LoadedLibraryName : string; Override
```

**Visibility:** default

**Description:** LoadedLibraryName returns the actually loaded library name.

**See also:** DefaultLibraryName (708)
CHAPTER 27. REFERENCE FOR UNIT 'MSSQLCNN'

27.7 TSybaseConnection

27.7.1 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)

27.7.2 Method overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>709</td>
<td>Create</td>
<td>Create a Sybase database connection</td>
</tr>
</tbody>
</table>

27.7.3 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.

27.8 TSybaseConnectionDef

27.8.1 Description
Describes the MS SQL connection properties for TSQLConnector (701)

See also: TSybaseConnection (709), TSQLConnector (701)

27.8.2 Method overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>710</td>
<td>ConnectionClass</td>
<td>Connection class to use.</td>
</tr>
<tr>
<td>710</td>
<td>Description</td>
<td>Short description of connection</td>
</tr>
<tr>
<td>709</td>
<td>TypeName</td>
<td>Connection type name</td>
</tr>
</tbody>
</table>

27.8.3 TSybaseConnectionDef.TypeName

Synopsis: Connection type name
27.8.4 TSybaseConnectionDef.ConnectionClass

Synopsis: Connection class to use.

Declaration: class function ConnectionClass : TSQLConnectionClass; Override

Visibility: default

Description: ConnectionClass returns TSybaseConnection (709)

See also: TSybaseConnection (709)

27.8.5 TSybaseConnectionDef.Description

Synopsis: Short description of connection

Declaration: class function Description : string; Override

Visibility: default

Description: Description describes the Sybase connector type.
Chapter 28

Reference for unit ’nullstream’

28.1 Used units

Table 28.1: Used units by unit ’nullstream’

<table>
<thead>
<tr>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Classes</td>
<td>??</td>
</tr>
<tr>
<td>System</td>
<td>??</td>
</tr>
</tbody>
</table>

28.2 Overview

The nullstream unit implements TNullStream (711), 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.

28.3 ENullStreamError

28.3.1 Description

ENullStreamError is the exception raised when TNullStream.Seek (712) results in an invalid position.

See also: TNullStream.Seek (712)

28.4 TNullStream

28.4.1 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 (712), TNullStream.Write (712), #rtl.classes.TStream.Size (??)
### 28.4.2 Method overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>713</td>
<td>Create</td>
<td>Create a new instance</td>
</tr>
<tr>
<td>712</td>
<td>Read</td>
<td>Read null bytes from the stream</td>
</tr>
<tr>
<td>712</td>
<td>Seek</td>
<td>Set current position in the stream</td>
</tr>
<tr>
<td>712</td>
<td>Write</td>
<td>Write to stream</td>
</tr>
</tbody>
</table>

### 28.4.3 TNullStream.Read

**Synopsis:** Read null bytes from the stream

**Declaration:**

```pascal
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 (712), #rtl.classes.TStream.Size (??)

### 28.4.4 TNullStream.Write

**Synopsis:** Write to stream

**Declaration:**

```pascal
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 (712), #rtl.classes.TStream.Size (??)

### 28.4.5 TNullStream.Seek

**Synopsis:** Set current position in the stream

**Declaration:**

```pascal
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 (711) exception is raised.

See also: TNullStream.Read (712), TNullStream.Write (712), #rtl.classes.TStream.Seek (??)
28.4.6 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 29

Reference for unit ’Pipes’

29.1 Used units

<table>
<thead>
<tr>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Classes</td>
<td>??</td>
</tr>
<tr>
<td>System</td>
<td>??</td>
</tr>
<tr>
<td>sysutils</td>
<td>??</td>
</tr>
</tbody>
</table>

29.2 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.

29.3 Constants, types and variables

29.3.1 Constants

ENoSeekMsg = 'Cannot seek on pipes'

Constant used in EPipeSeek (715) exception.

EPipeMsg = 'Failed to create pipe.'

Constant used in EPipeCreation (715) exception.

29.4 Procedures and functions

29.4.1 CreatePipeHandles

Synopsis: Function to create a set of pipe handles
CHAPTER 29. REFERENCE FOR UNIT 'PIPES'

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 (715)

29.4.2 CreatePipeStreams

Synopsis: Create a pair of pipe streams.

Declaration: procedure CreatePipeStreams(var InPipe: TInputPipeStream;
    var OutPipe: TOutputPipeStream)

Visibility: default

Description: CreatePipeStreams creates a set of pipe file descriptors with CreatePipeHandles (714), 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 (714) 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 (715) exception is raised.

See also: CreatePipeHandles (714), TInputPipeStream (716), TOutputPipeStream (718)

29.5 EPipeCreation

29.5.1 Description
Exception raised when an error occurred during the creation of a pipe pair.

29.6 EPipeError

29.6.1 Description
Exception raised when an invalid operation is performed on a pipe stream.

29.7 EPipeSeek

29.7.1 Description
Exception raised when an invalid seek operation is attempted on a pipe.

715
29.8  TInputPipeStream

29.8.1  Description

TInputPipeStream is created by the CreatePipeStreams (715) 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 (715), TOutputPipeStream (718)

29.8.2  Method overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>716</td>
<td>Destroy</td>
<td>Destroy this instance of the input pipe stream</td>
</tr>
<tr>
<td>717</td>
<td>Read</td>
<td>Read data from the stream to a buffer.</td>
</tr>
<tr>
<td>717</td>
<td>Seek</td>
<td>Set the current position of the stream</td>
</tr>
<tr>
<td>716</td>
<td>Write</td>
<td>Write data to the stream.</td>
</tr>
</tbody>
</table>

29.8.3  Property overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Properties</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>717</td>
<td>NumBytesAvailable</td>
<td>r</td>
<td>Number of bytes available for reading.</td>
</tr>
</tbody>
</table>

29.8.4  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 (716)

29.8.5  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 (717), Seek (717)
29.8.6 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 (715) exception is raised if the stream does not allow the requested seek operation.

See also: EPipeSeek (715), Seek (?)

29.8.7 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 (716), Seek (717)

29.8.8 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 (717)
CHAPTER 29. REFERENCE FOR UNIT 'PIPES'

29.9 TOutputPipeStream

29.9.1 Description
TOutputPipeStream is created by the CreatePipeStreams (715) call to represent the writing end of a pipe. It is a TStream (??) descendent which does not allow reading.

See also: TStream (??), CreatePipeStreams (715), TInputPipeStream (716)

29.9.2 Method overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>718</td>
<td>Destroy</td>
<td>Destroy this instance of the output pipe stream</td>
</tr>
<tr>
<td>718</td>
<td>Read</td>
<td>Read data from the stream.</td>
</tr>
<tr>
<td>718</td>
<td>Seek</td>
<td>Sets the position in the stream</td>
</tr>
</tbody>
</table>

29.9.3 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 (??)

29.9.4 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 (715) exception is raised if this method is called.

29.9.5 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 (718)
Chapter 30

Reference for unit ’pooledmm’

30.1 Used units

Table 30.1: Used units by unit ’pooledmm’

<table>
<thead>
<tr>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Classes</td>
<td>??</td>
</tr>
<tr>
<td>System</td>
<td>??</td>
</tr>
</tbody>
</table>

30.2 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.

30.3 Constants, types and variables

30.3.1 Types

PPooledMemManagerItem = ^TPooledMemManagerItem

PPooledMemManagerItem is a pointer type, pointing to a TPooledMemManagerItem (720) item, used in a linked list.

TEnumItemsMethod = procedure(Item: Pointer) of object

TEnumItemsMethod is a prototype for the callback used in the TNonFreePooledMemManager.EnumerateItems (721) call. The parameter Item will be set to each of the pointers in the item list of TNonFreePooledMemManager (720).

30.4 TPooledMemManagerItem

TPooledMemManagerItem = record

719
TPooledMemManagerItem is used internally by the TPooledMemManager (722) class to maintain the free list block. It simply points to the next free block.

30.5 TNonFreePooledMemManager

30.5.1 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 an 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.

30.5.2 Method overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>720</td>
<td>Clear</td>
<td>Clears the memory</td>
</tr>
<tr>
<td>720</td>
<td>Create</td>
<td>Creates a new instance of TNonFreePooledMemManager</td>
</tr>
<tr>
<td>721</td>
<td>Destroy</td>
<td>Removes the TNonFreePooledMemManager instance from memory</td>
</tr>
<tr>
<td>721</td>
<td>EnumerateItems</td>
<td>Enumerator all items in the list</td>
</tr>
<tr>
<td>721</td>
<td>NewItem</td>
<td>Return a pointer to a new memory block</td>
</tr>
</tbody>
</table>

30.5.3 Property overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Properties</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>721</td>
<td>ItemSize</td>
<td>r</td>
<td>Size of an item in the list</td>
</tr>
</tbody>
</table>

30.5.4 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 (721) is valid after a call to Clear

See also: NewItem (721)

30.5.5 TNonFreePooledMemManager.Create

Synopsis: Creates a new instance of TNonFreePooledMemManager

Declaration: constructor Create(TheItemSize: Integer)

Visibility: public
CHAPTER 30. REFERENCE FOR UNIT 'POOLEDMM'

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 (721)

30.5.6 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 (720), TNonFreePooledMemManager.Clear (720)

30.5.7 TNonFreePooledMemManagerNewItem

Synopsis: Return a pointer to a new memory block

Declaration: functionNewItem : Pointer

Visibility: public

Description:NewItem returns a pointer to an unused memory block of size ItemSize (721). 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 (720)

30.5.8 TNonFreePooledMemManagerEnumerateItems

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)

30.5.9 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 (720)
30.6 TPooledMemManager

30.6.1 Description
TPooledMemManager is a class which maintains a linked list of blocks, represented by the TPooledMemManagerItem (720) record. It should not be used directly, but should be descended from and the descendent should implement the actual memory manager.

See also: TPooledMemManagerItem (720)

30.6.2 Method overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>722</td>
<td>Clear</td>
<td>Clears the list</td>
</tr>
<tr>
<td>722</td>
<td>Create</td>
<td>Creates a new instance of the TPooledMemManager class</td>
</tr>
<tr>
<td>723</td>
<td>Destroy</td>
<td>Removes an instance of TPooledMemManager class from memory</td>
</tr>
</tbody>
</table>

30.6.3 Property overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Properties</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>724</td>
<td>AllocatedCount</td>
<td>r</td>
<td>Total number of allocated items in the list</td>
</tr>
<tr>
<td>723</td>
<td>Count</td>
<td>r</td>
<td>Number of items in the list</td>
</tr>
<tr>
<td>724</td>
<td>FreeCount</td>
<td>r</td>
<td>Number of free items in the list</td>
</tr>
<tr>
<td>724</td>
<td>FreedCount</td>
<td>r</td>
<td>Total number of freed items in the list</td>
</tr>
<tr>
<td>723</td>
<td>MaximumFreeCountRatio</td>
<td>rw</td>
<td>Maximum ratio of free items over total items</td>
</tr>
<tr>
<td>723</td>
<td>MinimumFreeCount</td>
<td>rw</td>
<td>Minimum count of free items in the list</td>
</tr>
</tbody>
</table>

30.6.4 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 (724)

30.6.5 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 (723)
30.6.6 TPooledMemManager.Destroy

Synopsis: Removes an instance of TPooledMemManager class from memory

Declaration: destructor Destroy; Override

Visibility: public

Description: Destroy calls Clear (722) and then calls the inherited destroy. Destroy should never be called directly. Instead Free should be used, or FreeAndNil

See also: TPooledMemManager.Create (722)

30.6.7 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 (723)

30.6.8 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 (723)

30.6.9 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 (724), TPooledMemManager.AllocatedCount (724), TPooledMemManager.FreedCount (724)
30.6.10 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 (723), TPooledMemManager.AllocatedCount (724), TPooledMemManager.FreedCount (724)

30.6.11 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 (723), TPooledMemManager.FreeCount (724), TPooledMemManager.FreedCount (724)

30.6.12 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 (723), TPooledMemManager.FreeCount (724), TPooledMemManager.AllocatedCount (724)
Chapter 31

Reference for unit ’process’

31.1 Used units

Table 31.1: Used units by unit ’process’

<table>
<thead>
<tr>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Classes</td>
<td>??</td>
</tr>
<tr>
<td>Math</td>
<td>??</td>
</tr>
<tr>
<td>Pipes</td>
<td>714</td>
</tr>
<tr>
<td>System</td>
<td>??</td>
</tr>
<tr>
<td>sysutils</td>
<td>??</td>
</tr>
</tbody>
</table>

31.2 Overview

The Process unit contains the code for the TProcess (731) 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.

31.3 Constants, types and variables

31.3.1 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 (731) class during the RunCommandLoop (736) 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 (741). 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 31.2: Enumeration values for type TProcessOption

<table>
<thead>
<tr>
<th>Value</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>poDebugOnlyThisProcess</td>
<td>Do not follow processes started by this process (Win32 only)</td>
</tr>
<tr>
<td>poDebugProcess</td>
<td>Allow debugging of the process (Win32 only)</td>
</tr>
<tr>
<td>poDefaultErrorMode</td>
<td>Use default error handling.</td>
</tr>
<tr>
<td>poDetached</td>
<td>Runs a process using the DETACHED_PROCESS creation flag on Windows</td>
</tr>
<tr>
<td>poNewConsole</td>
<td>Start a new console window for the process (Win32 only)</td>
</tr>
<tr>
<td>poNewProcessGroup</td>
<td>Start the process in a new process group (Win32 only)</td>
</tr>
<tr>
<td>poNoConsole</td>
<td>Do not allow access to the console window for the process (Win32 only)</td>
</tr>
<tr>
<td>poPassInput</td>
<td>Pass standard input handle on to new process</td>
</tr>
<tr>
<td>poRunIdle</td>
<td>Signals an event handler to wait for output in the run loop for a process.</td>
</tr>
<tr>
<td>poRunSuspended</td>
<td>Start the process in suspended state.</td>
</tr>
<tr>
<td>poStderrToOutput</td>
<td>Redirect standard error to the standard output stream.</td>
</tr>
<tr>
<td>poUsePipes</td>
<td>Use pipes to redirect standard input and output.</td>
</tr>
<tr>
<td>poWaitOnExit</td>
<td>Wait for the process to terminate before returning.</td>
</tr>
</tbody>
</table>

When a new process is started using TProcess.Execute (733), 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 (726).

TProcessPriority = (ppHigh, ppIdle, ppNormal, ppRealTime, ppBelowNormal, ppAboveNormal)
### Table 31.3: Enumeration values for type TProcessPriority

<table>
<thead>
<tr>
<th>Value</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>ppAboveNormal</td>
<td>Above normal priority</td>
</tr>
<tr>
<td>ppBelowNormal</td>
<td>Below normal priority</td>
</tr>
<tr>
<td>ppHigh</td>
<td>The process runs at higher than normal priority.</td>
</tr>
<tr>
<td>ppIdle</td>
<td>The process only runs when the system is idle (i.e. has nothing else to do)</td>
</tr>
<tr>
<td>ppNormal</td>
<td>The process runs at normal priority.</td>
</tr>
<tr>
<td>ppRealTime</td>
<td>The process runs at real-time priority.</td>
</tr>
</tbody>
</table>

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** (727) 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 31.4: Enumeration values for type TRunCommandEventCode

<table>
<thead>
<tr>
<th>Value</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>RunCommandException</td>
<td>An error happened during reading of the command</td>
</tr>
<tr>
<td>RunCommandFinished</td>
<td>The command finished</td>
</tr>
<tr>
<td>RunCommandIdle</td>
<td>No data was available for reading</td>
</tr>
<tr>
<td>RunCommandReadOutputString</td>
<td>Output from the command was read</td>
</tr>
<tr>
<td>RunCommandReadOutputStream</td>
<td>Output from the command was read as a string</td>
</tr>
</tbody>
</table>

**TRunCommandEventCode** is an enumerated type indicating the stage at which a process is during the **RunCommandLoop** (736) call, reported through the TProcess.OnRunCommandEvent (740) event handler.
TRunCommandEventCodeSet = Set of TRunCommandEventCode

TRunCommandEventCodeSet is a set of TRunCommandEventCode (727) values.

TShowWindowOptions = (swoNone, swoHIDE, swoMaximize, swoMinimize, swoRestore, swoShow, swoShowDefault, swoShowMaximized, swoShowMinimized, swoShowMinNOActive, swoShowNA, swoShowNoActivate, swoShowNormal)

<table>
<thead>
<tr>
<th>Value</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>swoHIDE</td>
<td>The main window is hidden.</td>
</tr>
<tr>
<td>swoMaximize</td>
<td>The main window is maximized.</td>
</tr>
<tr>
<td>swoMinimize</td>
<td>The main window is minimized.</td>
</tr>
<tr>
<td>swoNone</td>
<td>Allow system to position the window.</td>
</tr>
<tr>
<td>swoRestore</td>
<td>Restore the previous position.</td>
</tr>
<tr>
<td>swoShow</td>
<td>Show the main window.</td>
</tr>
<tr>
<td>swoShowDefault</td>
<td>When showing Show the main window on</td>
</tr>
<tr>
<td>swoShowMaximized</td>
<td>The main window is shown maximized</td>
</tr>
<tr>
<td>swoShowMinimized</td>
<td>The main window is shown minimized</td>
</tr>
<tr>
<td>swoShowMinNOActive</td>
<td>The main window is shown minimized but not activated</td>
</tr>
<tr>
<td>swoShowNA</td>
<td>The main window is shown but not activated</td>
</tr>
<tr>
<td>swoShowNoActivate</td>
<td>The main window is shown but not activated</td>
</tr>
<tr>
<td>swoShowNormal</td>
<td>The main window is shown normally</td>
</tr>
</tbody>
</table>

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>
<thead>
<tr>
<th>Value</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>suoUseCountChars</td>
<td>Use the console character width as specified in TProcess (731).</td>
</tr>
<tr>
<td>suoUseFillAttribute</td>
<td>Use the console fill attribute as specified in TProcess (731).</td>
</tr>
<tr>
<td>suoUsePosition</td>
<td>Use the window sizes as specified in TProcess (731).</td>
</tr>
<tr>
<td>suoUseShowWindow</td>
<td>Use the Show Window options specified in TShowWindowOption (728)</td>
</tr>
<tr>
<td>suoUseSize</td>
<td>Use the window sizes as specified in TProcess (731).</td>
</tr>
</tbody>
</table>

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 (728).
31.3.2 Variables

DefaultTProcess : TProcessClass = TPROCESS

DefaultTProcess is the process class used by the RunCommand (730) and RunCommandInDir (730) 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 (729) function. If XTermProgram (729) 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 (729) is called.

31.4 Procedures and functions

31.4.1 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 (742)

31.4.2 DetectXTerm

Synopsis: Detect the terminal program.

Declaration: function DetectXTerm : string

Visibility: default

Description: DetectXTerm checks if XTermProgram (729) is set. If so, it returns that. If XTermProgram is empty, the list specified in TryTerminals (729) 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 (729), TryTerminals (729), TProcess.XTermProgram (749)

### 31.4.3 RunCommand

**Synopsis:** Execute a command in the current working directory

**Declaration:**
```delphi
definition 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 (730) 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 (730)

### 31.4.4 RunCommandIndir

**Synopsis:** Run a command in a specific directory.

**Declaration:**
```delphi
definition 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 (726) 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 (731), RunCommand (730), TProcessOptions (726)

### 31.5 EProcess

**31.5.1 Description**

Exception raised when an error occurs in a TProcess routine.

See also: TProcess (731)

### 31.6 TPROCESS

**31.6.1 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 (742) property to the full pathname of the program that should be executed, and call Execute (733). To determine whether the process is still running (i.e. has not stopped executing), the Running (746) property can be checked.

More advanced techniques can be used with the Options (745) settings.

See also: Create (733), Execute (733), Running (746), CommandLine (742), Options (745)

### 31.6.2 Method overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>734</td>
<td>CloseInput</td>
<td>Close the input stream of the process</td>
</tr>
<tr>
<td>734</td>
<td>CloseOutput</td>
<td>Close the output stream of the process</td>
</tr>
<tr>
<td>734</td>
<td>CloseStderr</td>
<td>Close the error stream of the process</td>
</tr>
<tr>
<td>733</td>
<td>Create</td>
<td>Create a new instance of the TProcess class.</td>
</tr>
<tr>
<td>733</td>
<td>Destroy</td>
<td>Destroy this instance of TProcess</td>
</tr>
<tr>
<td>733</td>
<td>Execute</td>
<td>Execute the program with the given options</td>
</tr>
<tr>
<td>736</td>
<td>ReadInputStream</td>
<td>Read available data from input stream</td>
</tr>
<tr>
<td>734</td>
<td>Resume</td>
<td>Resume execution of a suspended process</td>
</tr>
<tr>
<td>736</td>
<td>RunCommandLoop</td>
<td>Execute command and collect output in strings</td>
</tr>
<tr>
<td>735</td>
<td>Suspend</td>
<td>Suspend a running process</td>
</tr>
<tr>
<td>735</td>
<td>Terminate</td>
<td>Terminate a running process</td>
</tr>
<tr>
<td>735</td>
<td>WaitOnExit</td>
<td>Wait for the program to stop executing.</td>
</tr>
</tbody>
</table>
# 31.6.3 Property overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Properties</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>741</td>
<td>Active</td>
<td>rw</td>
<td>Start or stop the process.</td>
</tr>
<tr>
<td>741</td>
<td>ApplicationName</td>
<td>rw</td>
<td>Name of the application to start (deprecated)</td>
</tr>
<tr>
<td>742</td>
<td>CommandLine</td>
<td>rw</td>
<td>Command-line to execute (deprecated)</td>
</tr>
<tr>
<td>743</td>
<td>ConsoleTitle</td>
<td>rw</td>
<td>Title of the console window</td>
</tr>
<tr>
<td>744</td>
<td>CurrentDirectory</td>
<td>rw</td>
<td>Working directory of the process.</td>
</tr>
<tr>
<td>744</td>
<td>Desktop</td>
<td>rw</td>
<td>Desktop on which to start the process.</td>
</tr>
<tr>
<td>744</td>
<td>Environment</td>
<td>rw</td>
<td>Environment variables for the new process</td>
</tr>
<tr>
<td>742</td>
<td>Executable</td>
<td>rw</td>
<td>Executable name. Supersedes CommandLine and ApplicationName.</td>
</tr>
<tr>
<td>740</td>
<td>ExitCode</td>
<td>r</td>
<td>Exit code of the process</td>
</tr>
<tr>
<td>739</td>
<td>ExitStatus</td>
<td>r</td>
<td>Exit status of the process</td>
</tr>
<tr>
<td>749</td>
<td>FillAttribute</td>
<td>rw</td>
<td>Color attributes of the characters in the console window (Windows only)</td>
</tr>
<tr>
<td>737</td>
<td>Handle</td>
<td>r</td>
<td>Handle of the process</td>
</tr>
<tr>
<td>740</td>
<td>InheritHandles</td>
<td>rw</td>
<td>Should the created process inherit the open handles of the current process.</td>
</tr>
<tr>
<td>738</td>
<td>Input</td>
<td>r</td>
<td>Stream connected to standard input of the process.</td>
</tr>
<tr>
<td>741</td>
<td>OnForkEvent</td>
<td>rw</td>
<td>Event triggered after fork occurred on Linux</td>
</tr>
<tr>
<td>740</td>
<td>OnRunCommandEvent</td>
<td>rw</td>
<td>Event handler, called when RunCommandLoop is executing</td>
</tr>
<tr>
<td>745</td>
<td>Options</td>
<td>rw</td>
<td>Options to be used when starting the process.</td>
</tr>
<tr>
<td>738</td>
<td>Output</td>
<td>r</td>
<td>Stream connected to standard output of the process.</td>
</tr>
<tr>
<td>741</td>
<td>PipeBufferSize</td>
<td>rw</td>
<td>Buffer size to be used when using pipes</td>
</tr>
<tr>
<td>745</td>
<td>Priority</td>
<td>rw</td>
<td>Priority at which the process is running</td>
</tr>
<tr>
<td>737</td>
<td>ProcessHandle</td>
<td>r</td>
<td>Alias for Handle (737)</td>
</tr>
<tr>
<td>737</td>
<td>ProcessID</td>
<td>r</td>
<td>ID of the process</td>
</tr>
<tr>
<td>740</td>
<td>RunCommandSleepTime</td>
<td>rw</td>
<td>Sleep time between attempts to collect data</td>
</tr>
<tr>
<td>746</td>
<td>Running</td>
<td>r</td>
<td>Determines whether the process is still running.</td>
</tr>
<tr>
<td>747</td>
<td>ShowWindow</td>
<td>rw</td>
<td>Determines how the process main window is shown (Windows only)</td>
</tr>
<tr>
<td>746</td>
<td>StartupOptions</td>
<td>rw</td>
<td>Additional (Windows) startup options</td>
</tr>
<tr>
<td>739</td>
<td>Stderr</td>
<td>r</td>
<td>Stream connected to standard diagnostic output of the process</td>
</tr>
<tr>
<td>737</td>
<td>ThreadHandle</td>
<td>r</td>
<td>Main process thread handle</td>
</tr>
<tr>
<td>738</td>
<td>ThreadID</td>
<td>r</td>
<td>ID of the main process</td>
</tr>
<tr>
<td>747</td>
<td>WindowColumns</td>
<td>rw</td>
<td>Number of columns in console window (windows only)</td>
</tr>
<tr>
<td>747</td>
<td>WindowHeight</td>
<td>rw</td>
<td>Height of the process main window</td>
</tr>
<tr>
<td>748</td>
<td>WindowLeft</td>
<td>rw</td>
<td>X-coordinate of the initial window (Windows only)</td>
</tr>
<tr>
<td>736</td>
<td>WindowRect</td>
<td>rw</td>
<td>Positions for the main program window.</td>
</tr>
<tr>
<td>748</td>
<td>WindowRows</td>
<td>rw</td>
<td>Number of rows in console window (Windows only)</td>
</tr>
<tr>
<td>748</td>
<td>WindowTop</td>
<td>rw</td>
<td>Y-coordinate of the initial window (Windows only)</td>
</tr>
<tr>
<td>749</td>
<td>WindowWidth</td>
<td>rw</td>
<td>Height of the process main window (Windows only)</td>
</tr>
<tr>
<td>749</td>
<td>XTermProgram</td>
<td>rw</td>
<td>XTerm program to use (UNIX only)</td>
</tr>
</tbody>
</table>
31.6.4 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.

31.6.5 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 (733)

31.6.6 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 (742), applying as much as of the specified options as supported on the current platform.

If the poWaitOnExit option is specified in Options (745), 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 (735) call can be used to wait for it to close, or the Running (746) call can be used to check whether it is still running.

The TProcess.Terminate (735) call can be used to terminate the program if it is still running, or the Suspend (735) call can be used to temporarily stop the program’s execution.

The ExitStatus (739) function can be used to check the program’s exit status, after it has stopped executing.

Errors: On error a EProcess (731) exception is raised.

31.6.7 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 (738), StdErr (739), Output (738), CloseOutput (734), CloseStdErr (734)

31.6.8 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 (738), Input (738), StdErr (739), CloseInput (734), CloseStdErr (734)

31.6.9 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 (738), Input (738), StdErr (739), CloseInput (734), CloseStdErr (734)

31.6.10 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 (745).

Errors: None.

See also: TProcess.Suspend (735), TProcess.Options (745), TProcess.Execute (733), TProcess.Terminate (735)
31.6.11 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 (734) call.

Suspend is fundamentally different from TProcess.Terminate (735) which actually stops the process.

Errors: On error, a nonzero result is returned.

See also: TProcess.Options (745), TProcess.Resume (734), TProcess.Terminate (735), TProcess.Execute (733)

31.6.12 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 (739), TProcess.Suspend (735), TProcess.Execute (733), TProcess.WaitOnExit (735), TProcess.ExitCode (740)

31.6.13 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 (739), TProcess.Terminate (735), TProcess.Running (746), TProcess.ExitCode (740)
31.6.14 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 successfully 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.

31.6.15 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 (740) event handler is called during the various stages of the call. If it is not explicitly set, a sleep period specified by TProcess.RunCommandSleepTime (740) is interjected between the various read calls.

The return value of this call is 1 for error, zero for success.

See also: TProcess.OnRunCommandEvent (740), TProcess.RunCommandSleepTime (740)

31.6.16 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...
### 31.6.17 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 (733) has been called. It is not reset after the process stopped.

**See also:** TProcess.ThreadHandle (737), TProcess.ProcessID (737), TProcess.ThreadID (738)

### 31.6.18 TPROCESS.ProcessHandle

**Synopsis:** Alias for Handle (737)

**Declaration:**
```
Property ProcessHandle : THandle
```

**Visibility:** public  
**Access:** Read

**Description:** ProcessHandle equals Handle (737) and is provided for completeness only.

**See also:** TProcess.Handle (737), TProcess.ThreadHandle (737), TProcess.ProcessID (737), TProcess.ThreadID (738)

### 31.6.19 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 (733) has been called. It is not reset after the process stopped.

**See also:** TProcess.Handle (737), TProcess.ProcessID (737), TProcess.ThreadID (738)

### 31.6.20 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 (733) has been called. It is not reset after the process stopped.

See also: TProcess.Handle (737), TProcess.ThreadHandle (737), TProcess.ThreadID (738)

31.6.21 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 (733) has been called. It is not reset after the process stopped.

See also: TProcess.ProcessID (737), TProcess.Handle (737), TProcess.ThreadHandle (737)

31.6.22 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 (745).

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 (738), TProcess.StdErr (739), TProcess.Options (745), TProcessOption (726)

31.6.23 TPROCESS.Output

Synopsis: Stream connected to standard output of the process.

Declaration: Property Output : TInputPipeStream

Visibility: public
Access: Read
CHAPTER 31. REFERENCE FOR UNIT 'PROCESS'

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 (745).

The Output stream also contains any data written to standard diagnostic output (stderr) when the poStdErrToOutPut flag is used in Options (745).

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 (738), TProcess.StdErr (739), TProcess.Options (745), TProcessOption (726)

31.6.24 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 instatiated when the poUsePipes flag is used in Options (745).

The Output stream equals the Output (738) when the poStdErrToOutPut flag is used in Options (745).

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 (738), TProcess.Output (738), TProcess.Options (745), TProcessOption (726)

31.6.25 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 (746), TProcess.Terminate (735), TProcess.ExitCode (740)
31.6.26 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 (725) 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 (739), TProcess.Running (746), TProcess.WaitOnExit (735), TProcess.Terminate (735)

31.6.27 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 (738), TProcess.Output (738), TProcess.StdErr (739)

31.6.28 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 (736) call and when an exception occurs.

See also: TProcess.RunCommandLoop (736)

31.6.29 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
31.6.30 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 (738), Input (738), StdErr (739), CloseInput (734), CloseStdErr (734), TProcessForkEvent (726)

31.6.31 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 (714)

31.6.32 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 (733), TProcess.Terminate (735)

31.6.33 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 (742). 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 (742) instead, and leave ApplicationName empty.

See also: TProcess.CommandLine (742), TProcess.Executable (742), TProcess.Parameters (743)

31.6.34 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 (742) and TProcess.Parameters (743), 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 (741), TProcess.Executable (742), TProcess.Parameters (743)

31.6.35 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 (743).

Executable supersedes the TProcess.CommandLine (742) and TProcess.ApplicationName (741) properties, which have been deprecated. However, if either of CommandLine or ApplicationName is specified, they will be used instead of Executable.

See also: CommandLine (742), ApplicationName (741), Parameters (743)
31.6.36 TPROCESS.Parameters


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 (742).

Command-line arguments should be specified one per item in Parameters: each item in Parameters will be passed as a separate command-line item. It is therefore 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 (742), CommandLine (742), ApplicationName (741)

31.6.37 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 (747), TProcess.WindowRows (748)
31.6.38 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 (744)

31.6.39 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 (738), TProcess.Output (738), TProcess.StdErr (739)

31.6.40 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 (745)
31.6.41 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 (733) call is made.

Table 31.7:

<table>
<thead>
<tr>
<th>Option</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>poRunSuspended</td>
<td>Start the process in suspended state.</td>
</tr>
<tr>
<td>poWaitOnExit</td>
<td>Wait for the process to terminate before returning.</td>
</tr>
<tr>
<td>poUsePipes</td>
<td>Use pipes to redirect standard input and output.</td>
</tr>
<tr>
<td>poStderrToOutput</td>
<td>Redirect standard error to the standard output stream.</td>
</tr>
<tr>
<td>poNoConsole</td>
<td>Do not allow access to the console window for the process (Win32 only)</td>
</tr>
<tr>
<td>poNewConsole</td>
<td>Start a new console window for the process (Win32 only)</td>
</tr>
<tr>
<td>poDefaultErrorMode</td>
<td>Use default error handling.</td>
</tr>
<tr>
<td>poNewProcessGroup</td>
<td>Start the process in a new process group (Win32 only)</td>
</tr>
<tr>
<td>poDebugProcess</td>
<td>Allow debugging of the process (Win32 only)</td>
</tr>
<tr>
<td>poDebugOnlyThisProcess</td>
<td>Do not follow processes started by this process (Win32 only)</td>
</tr>
</tbody>
</table>

See also: TProcessOption (726), TProcessOptions (726), TProcess.Priority (745), TProcess.StartUpOptions (746)

31.6.42 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 31.8:

<table>
<thead>
<tr>
<th>Priority</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>ppHigh</td>
<td>The process runs at higher than normal priority.</td>
</tr>
<tr>
<td>ppIdle</td>
<td>The process only runs when the system is idle (i.e. has nothing else to do)</td>
</tr>
<tr>
<td>ppNormal</td>
<td>The process runs at normal priority.</td>
</tr>
<tr>
<td>ppRealTime</td>
<td>The process runs at real-time priority.</td>
</tr>
</tbody>
</table>

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>
<thead>
<tr>
<th>Priority</th>
<th>Nice value</th>
</tr>
</thead>
<tbody>
<tr>
<td>ppHigh</td>
<td>20</td>
</tr>
<tr>
<td>ppIdle</td>
<td>20</td>
</tr>
<tr>
<td>ppNormal</td>
<td>0</td>
</tr>
<tr>
<td>ppRealTime</td>
<td>-20</td>
</tr>
</tbody>
</table>

See also: TProcessPriority (726)

### 31.6.43 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>
<thead>
<tr>
<th>Priority</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>suoUseShowWindow</td>
<td>Use the Show Window options specified in ShowWindow (747)</td>
</tr>
<tr>
<td>suoUseSize</td>
<td>Use the specified window sizes</td>
</tr>
<tr>
<td>suoUsePosition</td>
<td>Use the specified window sizes.</td>
</tr>
<tr>
<td>suoUseCountChars</td>
<td>Use the specified console character width.</td>
</tr>
<tr>
<td>suoUseFillAttribute</td>
<td>Use the console fill attribute specified in FillAttribute (749).</td>
</tr>
</tbody>
</table>

See also: TProcess.ShowWindow (747), TProcess.WindowHeight (747), TProcess.WindowWidth (749), TProcess.WindowLeft (748), TProcess.WindowTop (748), TProcess.WindowColumns (747), TProcess.WindowRows (748), TProcess.FillAttribute (749)

### 31.6.44 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 (735), TProcess.Active (741), TProcess.ExitStatus (739), TProcess.ExitCode (740)
31.6.45 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>
<thead>
<tr>
<th>Option</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>swoNone</td>
<td>Allow system to position the window.</td>
</tr>
<tr>
<td>swoHIDE</td>
<td>The main window is hidden.</td>
</tr>
<tr>
<td>swoMaximize</td>
<td>The main window is maximized.</td>
</tr>
<tr>
<td>swoMinimize</td>
<td>The main window is minimized.</td>
</tr>
<tr>
<td>swoRestore</td>
<td>Restore the previous position.</td>
</tr>
<tr>
<td>swoShow</td>
<td>Show the main window.</td>
</tr>
<tr>
<td>swoShowDefault</td>
<td>When showing Show the main window on a default position</td>
</tr>
<tr>
<td>swoShowMaximized</td>
<td>The main window is shown maximized</td>
</tr>
<tr>
<td>swoShowMinimized</td>
<td>The main window is shown minimized</td>
</tr>
<tr>
<td>swoShowMinNOActive</td>
<td>The main window is shown minimized but not activated</td>
</tr>
<tr>
<td>swoShowNormal</td>
<td>The main window is shown normally</td>
</tr>
</tbody>
</table>

31.6.46 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 (746)

See also: TProcess.WindowHeight (747), TProcess.WindowWidth (749), TProcess.WindowLeft (748), TProcess.WindowTop (748), TProcess.WindowRows (748), TProcess.FillAttribute (749), TProcess.StartupOptions (746)

31.6.47 TPROCESS.WindowHeight

Synopsis: Height of the process main window

Declaration: Property WindowHeight : Cardinal

Visibility: published

Access: Read, Write
CHAPTER 31. REFERENCE FOR UNIT 'PROCESS'

Description: WindowHeight is the initial height (in pixels) of the process’ main window. This property is only effective if suoUseSize is specified in StartupOptions (746).

See also: TProcess.WindowWidth (749), TProcess.WindowLeft (748), TProcess.WindowTop (748), TProcess.WindowColumns (747), TProcess.WindowRows (748), TProcess.FillAttribute (749), TProcess.StartupOptions (746)

31.6.48 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 (746).

See also: TProcess.WindowHeight (747), TProcess.WindowWidth (749), TProcess.WindowTop (748), TProcess.WindowColumns (747), TProcess.WindowRows (748), TProcess.FillAttribute (749), TProcess.StartupOptions (746)

31.6.49 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 (746).

See also: TProcess.WindowHeight (747), TProcess.WindowWidth (749), TProcess.WindowLeft (748), TProcess.WindowTop (748), TProcess.WindowColumns (747), TProcess.FillAttribute (749), TProcess.StartupOptions (746)

31.6.50 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 (746).

See also: TProcess.WindowHeight (747), TProcess.WindowWidth (749), TProcess.WindowLeft (748), TProcess.WindowColumns (747), TProcess.FillAttribute (749), TProcess.StartupOptions (746)

748
31.6.51  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 (746)

See also: TProcess.WindowHeight (747), TProcess.WindowLeft (748), TProcess.WindowTop (748), TProcess.WindowColumns (747), TProcess.WindowRows (748), TProcess.FillAttribute (749), TProcess.StartupOptions (746)

31.6.52  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 (747), TProcess.WindowWidth (749), TProcess.WindowLeft (748), TProcess.WindowTop (748), TProcess.WindowColumns (747), TProcess.WindowRows (748), TProcess.StartupOptions (746)

31.6.53  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 (745).

If none is specified, DetectXTerm (729) 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 (745), DetectXTerm (729)
Chapter 32

Reference for unit ’RttiUtils’

32.1 Used units

Table 32.1: Used units by unit ’RttiUtils’

<table>
<thead>
<tr>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Classes</td>
<td>??</td>
</tr>
<tr>
<td>StrUtils</td>
<td>??</td>
</tr>
<tr>
<td>System</td>
<td>??</td>
</tr>
<tr>
<td>sysutils</td>
<td>??</td>
</tr>
<tr>
<td>TypInfo</td>
<td>??</td>
</tr>
</tbody>
</table>

32.2 Overview

The rttiutils unit is a unit providing simplified access to the RTTI information from published properties using the TPropInfoList (753) 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 (755) class can be used for this. The implementation is based on the apputils unit from RXLib by AO ROSNO and Master-Bank.

32.3 Constants, types and variables

32.3.1 Constants

sPropNameDelimiter : string = '_'

Separator used when constructing section/key names.

32.3.2 Types

TERaseSectEvent = procedure(const ASection: string) of object
TeraseSectEvent is used by TPropsStorage (755) 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>
<thead>
<tr>
<th>Value</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>psoAlwaysStoreStringsCount</td>
<td>Always store the count of strings. Default is not to store the count</td>
</tr>
</tbody>
</table>

TPropStorageOption is the enumeration type used in the TPropsStorage.Options (758) property of TPropsStorage (755).

TPropStorageOptions = Set of TPropStorageOption

TPropStorageOptions is the set of TPropStorageOption used in TPropsStorage.Options (758).

TReadStrEvent = function(const ASection: string; const Item: string ;
                        const Default: string) : string of object

TReadStrEvent is used by TPropsStorage (755) 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 (755) 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.

### 32.3.3 Variables

FindGlobalComponentCallBack : TFindComponentEvent

FindGlobalComponentCallBack is called by UpdateStoredList (752) 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.
32.4 Procedures and functions

32.4.1 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 (752), TPropsStorage.StoreObjectsProps (757) or TPropsStorage.LoadObjectsProps (757).

See also: ParseStoredItem (752), UpdateStoredList (752), TPropsStorage.StoreObjectsProps (757), TPropsStorage.LoadObjectsProps (757)

32.4.2 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 (752). 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 (752), UpdateStoredList (752), TPropsStorage.StoreObjectsProps (757), TPropsStorage.LoadObjectsProps (757)

32.4.3 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 (752) 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 (751) 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 (752), TPropsStorage.StoreObjectsProps (757), TPropsStorage.LoadObjectsProps (757), FindGlobalComponentCallback (751)

32.5 TPropInfoList

32.5.1 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 (755)

See also: TPropsStorage (755)

32.5.2 Method overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>754</td>
<td>Contains</td>
<td>Check whether a certain property is included</td>
</tr>
<tr>
<td>753</td>
<td>Create</td>
<td>Create a new instance of TPropInfoList</td>
</tr>
<tr>
<td>754</td>
<td>Delete</td>
<td>Delete property information from the list</td>
</tr>
<tr>
<td>753</td>
<td>Destroy</td>
<td>Remove the TPropInfoList instance from memory</td>
</tr>
<tr>
<td>754</td>
<td>Find</td>
<td>Retrieve property information based on name</td>
</tr>
<tr>
<td>754</td>
<td>Intersect</td>
<td>Intersect 2 property lists</td>
</tr>
</tbody>
</table>

32.5.3 Property overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Properties</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>755</td>
<td>Count</td>
<td>r</td>
<td>Number of items in the list</td>
</tr>
<tr>
<td>755</td>
<td>Items</td>
<td>r</td>
<td>Indexed access to the property type pointers</td>
</tr>
</tbody>
</table>

32.5.4 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 (753)

32.5.5 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.
32.6 **TPropInfoList.Contains**

**Synopsis:** Check whether a certain property is included

**Declaration:**
```plaintext
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 (754), Intersect (754)

32.7 **TPropInfoList.Find**

**Synopsis:** Retrieve property information based on name

**Declaration:**
```plaintext
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 (754), Contains (754)

32.8 **TPropInfoList.Delete**

**Synopsis:** Delete property information from the list

**Declaration:**
```plaintext
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 (754) call.

**Errors:** No checking on the validity of \( Index \) is performed.

**See also:** Intersect (754)

32.9 **TPropInfoList.Intersect**

**Synopsis:** Intersect 2 property lists

**Declaration:**
```plaintext
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 (754), Contains (754)
32.5.10 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 (755)

32.5.11 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 (755)

32.6 TPropsStorage

32.6.1 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 (753)

32.6.2 Method overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>756</td>
<td>LoadAnyProperty</td>
<td>Load a property value</td>
</tr>
<tr>
<td>757</td>
<td>LoadObjectsProps</td>
<td>Load a list of component properties</td>
</tr>
<tr>
<td>757</td>
<td>LoadProperties</td>
<td>Load a list of properties</td>
</tr>
<tr>
<td>756</td>
<td>StoreAnyProperty</td>
<td>Store a property value</td>
</tr>
<tr>
<td>757</td>
<td>StoreObjectsProps</td>
<td>Store a list of component properties</td>
</tr>
<tr>
<td>756</td>
<td>StoreProperties</td>
<td>Store a list of properties</td>
</tr>
</tbody>
</table>
32.6.3 Property overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Properties</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>758</td>
<td>AObject</td>
<td>rw</td>
<td>Object to load or store properties from</td>
</tr>
<tr>
<td>760</td>
<td>OnEraseSection</td>
<td>rw</td>
<td>Erase a section in storage</td>
</tr>
<tr>
<td>759</td>
<td>OnReadString</td>
<td>rw</td>
<td>Read a string value from storage</td>
</tr>
<tr>
<td>759</td>
<td>OnWriteString</td>
<td>rw</td>
<td>Write a string value to storage</td>
</tr>
<tr>
<td>758</td>
<td>Options</td>
<td>rw</td>
<td>Options to take into account when saving or loading properties from the storage</td>
</tr>
<tr>
<td>759</td>
<td>Prefix</td>
<td>rw</td>
<td>Prefix to use in storage</td>
</tr>
<tr>
<td>759</td>
<td>Section</td>
<td>rw</td>
<td>Section name for storage</td>
</tr>
</tbody>
</table>

32.6.4 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 (758) property of TPropsStorage.

Errors: If the property pointer is invalid or AObject is invalid, an exception will be raised.

See also: AObject (758), LoadAnyProperty (756), LoadProperties (757), StoreProperties (756)

32.6.5 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 (758) property of TPropsStorage.

Errors: If the property pointer is invalid or AObject is invalid, an exception will be raised.

See also: AObject (758), StoreAnyProperty (756), LoadProperties (757), StoreProperties (756)

32.6.6 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 (758) object.

Errors: If an invalid property name is specified, an exception will be raised.

See also: AObject (758), StoreAnyProperty (756), LoadProperties (757), LoadAnyProperty (756)
32.6.7 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 (758) object.

Errors: If an invalid property name is specified, an exception will be raised.

See also: AObject (758), StoreAnyProperty (756), StoreProperties (756), LoadAnyProperty (756)

32.6.8 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 (752) 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 AComponent should be the form component that owns the button and checkbox.

Note that this call removes the value of the AObject (758) property.

Errors: If an invalid component is specified, an exception will be raised.

See also: UpdateStoredList (752), StoreObjectsProps (757), LoadProperties (757), LoadAnyProperty (756)

32.6.9 TPropsStorage.StoreObjectsProps

Synopsis: Store a list of component properties

Declaration: procedure StoreObjectsProps(AComponent: TComponent;
                                      StoredList: TStrings)

Visibility: public
**CHAPTER 32. REFERENCE FOR UNIT 'RTTIUTILS'**

**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 (752) 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 AComponent should be the form component that owns the button and checkbox.

Note that this call removes the value of the AObject (758) property.

See also: UpdateStoredList (752), LoadObjectsProps (757), LoadProperties (757), LoadAnyProperty (756)

---

### 32.6.10 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:

- **psoAlwaysStoreStringsCount** Always store the count of strings. Default is not to store the count

See also: TPropStorageOptions (751), TPropStorageOption (751)

---

### 32.6.11 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 (757) or LoadObjectProps (757) will destroy any value that this property might have.

See also: LoadProperties (757), LoadAnyProperty (756), StoreProperties (756), StoreAnyProperty (756), StoreObjectProps (757), LoadObjectProps (757)
32.6.12  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 (759)

32.6.13  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 (759)

32.6.14  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 (759), OnEraseSection (760), TReadStrEvent (751)

32.6.15  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 (759), OnEraseSection (760), TWriteStrEvent (751)

759
32.6.16  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 (759), OnWriteString (759), TEraseSectEvent (750)
Chapter 33

Reference for unit ’simpleipc’

33.1 Used units

Table 33.1: Used units by unit ’simpleipc’

<table>
<thead>
<tr>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Classes</td>
<td>??</td>
</tr>
<tr>
<td>Contnrs</td>
<td>201</td>
</tr>
<tr>
<td>syncobjs</td>
<td>894</td>
</tr>
<tr>
<td>System</td>
<td>??</td>
</tr>
<tr>
<td>sysutils</td>
<td>??</td>
</tr>
</tbody>
</table>

33.2 Overview

The SimpleIPC unit provides classes to implement a simple, one-way IPC mechanism using string messages. It provides a TSimpleIPCServer (778) component for the server, and a TSimpleIPCClient (775) 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.
33.3 Constants, types and variables

33.3.1 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

33.3.2 Constants

MsgVersion = 1

Current version of the messaging protocol

mtString = 1

String message type

mtUnknown = 0

Unknown message type

33.3.3 Types

TIPCClientCommClass = Class of TIPCClientComm

TIPCClientCommClass is used by TSimpleIPCClient (775) to decide which kind of communication channel to set up.
TIPCMessageOverflowAction = (ipcmoaNone, ipcmoaDiscardOld, ipcmoaDiscardNew, ipcmoaError)

Table 33.2: Enumeration values for type TIPCMessageOverflowAction

<table>
<thead>
<tr>
<th>Value</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>ipcmoaDiscardNew</td>
<td>Discard the new message</td>
</tr>
<tr>
<td>ipcmoaDiscardOld</td>
<td>Discard the oldest message</td>
</tr>
<tr>
<td>ipcmoaError</td>
<td>Raise an error</td>
</tr>
<tr>
<td>ipcmoaNone</td>
<td>Do nothing, just add the message</td>
</tr>
</tbody>
</table>

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 (778) 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.

### 33.3.4 Variables

DefaultIPCClientClass : TIPCClientCommClass = Nil

DefaultIPCClientClass is filled with a class pointer indicating which kind of communication protocol class should be instantiated by the TSimpleIPCClient (775) 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 (778) 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.

### 33.4 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.

### 33.5 EIPCError

#### 33.5.1 Description

EIPCError is the exception used by the various classes in the SimpleIPC unit to report errors.

### 33.6 TIPCClientComm

#### 33.6.1 Description

TIPCClientComm is an abstract component which implements the client-side communication proto-
col. The behaviour expected of this class must be implemented in a platform-dependent descendent
class.

The TSimpleIPCClient (775) 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 (767) component.
The descendent components must always be implemented in pairs.

See also: TSimpleIPCClient (775), TIPCServerComm (767), TSimpleIPCServer (778)
33.6.2 Method overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>765</td>
<td>Connect</td>
<td>Connect to the server</td>
</tr>
<tr>
<td>765</td>
<td>Create</td>
<td>Create a new instance of the TIPCClientComm</td>
</tr>
<tr>
<td>765</td>
<td>Disconnect</td>
<td>Disconnect from the server</td>
</tr>
<tr>
<td>766</td>
<td>SendMessage</td>
<td>Send a message</td>
</tr>
<tr>
<td>766</td>
<td>ServerRunning</td>
<td>Check if the server is running.</td>
</tr>
</tbody>
</table>

33.6.3 Property overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Properties</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>766</td>
<td>Owner</td>
<td>r</td>
<td>TSimpleIPCClIENT instance for which communication must be handled.</td>
</tr>
</tbody>
</table>

33.6.4 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 (775) instance for which it will handle communication. It can be retrieved later using the Owner (766) property.

See also: Owner (766), TSimpleIPCClIENT (775)

33.6.5 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 (774) and ServerInstance (777) properties of the owning TSimpleIPCClIENT (775) instance.

Connect is called by the TSimpleIPCClIENT.Connect (776) call or when the Active (774) 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 (776), Active (774), Disconnect (765)

33.6.6 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 (776) call or when the Active (774) 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 (776), Active (774), Connect (765)

33.6.7 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 (775) component.

See also: ServerID (774), InstanceID (782)

33.6.8 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.

33.6.9 TIPCClientComm.Owner
Synopsis: TSimpleIPCClient instance for which communication must be handled.

Declaration: Property Owner : TSimpleIPCClient

Visibility: public

Access: Read

Description: Owner is the TSimpleIPCClient (775) 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 (775), TIPCClientComm.Create (765)
33.7 TIPCServerComm

33.7.1 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 descendant class.

The TSimpleIPCServer (778) class does not implement the messaging protocol by itself. Instead, it creates an instance of a (platform dependent) descendant of TIPCServerComm which handles the internals of the communication protocol.

The client side of the messaging protocol is handled by the TIPCClientComm (764) component. The descendant components must always be implemented in pairs.

See also: TSimpleIPCServer (778), TIPCClientComm (764)

33.7.2 Method overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>767</td>
<td>Create</td>
<td>Create a new instance of the communication handler</td>
</tr>
<tr>
<td>768</td>
<td>PeekMessage</td>
<td>See if a message is available.</td>
</tr>
<tr>
<td>768</td>
<td>ReadMessage</td>
<td>Read message from the channel.</td>
</tr>
<tr>
<td>767</td>
<td>StartServer</td>
<td>Start the server-side of the communication channel</td>
</tr>
<tr>
<td>768</td>
<td>StopServer</td>
<td>Stop the server side of the communication channel</td>
</tr>
</tbody>
</table>

33.7.3 Property overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Properties</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>769</td>
<td>InstanceID</td>
<td>r</td>
<td>Unique identifier for the communication channel.</td>
</tr>
<tr>
<td>769</td>
<td>Owner</td>
<td>r</td>
<td>TSimpleIPCServer instance for which to handle transport</td>
</tr>
</tbody>
</table>

33.7.4 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 (769) property.

See also: Owner (769)

33.7.5 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 (774) property of the TSimpleIPCServer (778) 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 (785)). The arrival of new messages can be acted upon with TSimpleIPCServer.OnMessageQueued (783).

Errors: In case of an error, an EIPCError (764) exception is raised.

See also: TSimpleIPCServer (778), TSimpleIPC.Active (774), TSimpleIPCServer.OnMessageQueued (783), TSimpleIPCServer.ThreadTimeOut (785)

### 33.7.6 TIPCServerComm.StopServer

**Synopsis:** Stop the server side of the communication channel.

**Declaration:**

```pascal
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 (774) property of the TSimpleIPCServer (778) instance is set to False.

Errors: In case of an error, an EIPCError (764) exception is raised.

See also: TSimpleIPCServer (778), TSimpleIPC.Active (774)

### 33.7.7 TIPCServerComm.PeekMessage

**Synopsis:** See if a message is available.

**Declaration:**

```pascal
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 (768) call.

See also: ReadMessage (768)

### 33.7.8 TIPCServerComm.ReadMessage

**Synopsis:** Read message from the channel.

**Declaration:**

```pascal
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 (768) to see if a message is available.

See also: TSimpleIPCServer (778)
33.7.9 TIPCServerComm.Owner

Synopsis: TSimpleIPCServer instance for which to handle transport

Declaration: Property Owner : TSimpleIPCServer

Visibility: public
Access: Read

Description: Owner refers to the TSimpleIPCServer (778) instance for which this instance of TSimpleIPCServer handles the transport. It is specified when the TIPCServerComm is created.

See also: TSimpleIPCServer (778)

33.7.10 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.

33.8 TIPCServerMsg

33.8.1 Description

TIPCServerMsg is an auxiliary class used in the IPC server class TSimpleIPCServer (778). It keeps the data for 1 message. The set of messages is managed in TIPCServerMsgQueue (771). There should normally be no need to use this class directly.

See also: TIPCServerMsgQueue (771)

33.8.2 Method overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>770</td>
<td>Create</td>
<td>Create a new instance of a server message</td>
</tr>
<tr>
<td>770</td>
<td>Destroy</td>
<td>Destroy an instance of a server message</td>
</tr>
</tbody>
</table>

33.8.3 Property overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Properties</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>770</td>
<td>MsgType</td>
<td>rw</td>
<td>Message type</td>
</tr>
<tr>
<td>771</td>
<td>OwnsStream</td>
<td>rw</td>
<td>Does the message own the stream</td>
</tr>
<tr>
<td>770</td>
<td>Stream</td>
<td>r</td>
<td>Stream to store message data</td>
</tr>
<tr>
<td>771</td>
<td>StringMessage</td>
<td>r</td>
<td>String message sent by client</td>
</tr>
</tbody>
</table>
33.8.4 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 (770)

33.8.5 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 (770)

33.8.6 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 (770)

33.8.7 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 (770)
33.8.8  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 (770)

33.8.9  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.

33.9  TIPCServerMsgQueue

33.9.1  Description

TIPCServerMsgQueue implements a message queue with FIFO characteristics. It has support for a maximum queue length (TIPCServerMsgQueue.MaxCount (773)) and various ways of dealing with overflowing queue (TIPCServerMsgQueue.MaxAction (773))

See also: TIPCServerMsgQueue.MaxCount (773), TIPCServerMsgQueue.MaxAction (773)

33.9.2  Method overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>772</td>
<td>Clear</td>
<td>Clear the message queue</td>
</tr>
<tr>
<td>772</td>
<td>Create</td>
<td>Create a new message queue instance</td>
</tr>
<tr>
<td>772</td>
<td>Destroy</td>
<td>Destroy server message queue instance</td>
</tr>
<tr>
<td>773</td>
<td>Pop</td>
<td>Remove the oldest message from the queue</td>
</tr>
<tr>
<td>772</td>
<td>Push</td>
<td>Add a new message to the queue</td>
</tr>
</tbody>
</table>

33.9.3  Property overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Properties</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>773</td>
<td>Count</td>
<td>r</td>
<td>Number of messages in the queue</td>
</tr>
<tr>
<td>773</td>
<td>MaxAction</td>
<td>rw</td>
<td>Action to take when the number of messages will exceed MaxCount.</td>
</tr>
<tr>
<td>773</td>
<td>MaxCount</td>
<td>rw</td>
<td>Maximum number of messages in the queue, 0 for unlimited</td>
</tr>
</tbody>
</table>
### 33.9.4 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 (773) and TIPCServerMsgQueue.MaxAction (773) with their default values (DefaultIPCMessageQueueLimit (764) and DefaultIPCMessageOverflowAction (763), respectively)

Note that the messages are owned by the queue till they are popped of the queue.

See also: TIPCServerMsgQueue.MaxCount (773), TIPCServerMsgQueue.MaxAction (773), DefaultIPCMessageQueueLimit (764), DefaultIPCMessageOverflowAction (763)

### 33.9.5 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 (772), TIPCServerMsgQueue.Clear (772)

### 33.9.6 TIPCServerMsgQueue.Clear

**Synopsis:** Clear the message queue

**Declaration:**
```
procedure Clear;
```

**Visibility:** public

**Description:**
Clear discards the remaining messages in the list.

### 33.9.7 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 (773)) 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 (773) is set to ipcmoaError, an exception will be raised.

See also: TIPCServerMsgQueue.MaxCount (773), TIPCServerMsgQueue.MaxAction (773), TIPCServerMsgQueue.Pop (773)
### 33.9.8 TIPCServerMsgQueue.Pop

**Synopsis:** Remove the oldest message from the queue

**Declaration:**
```pascal
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 (772)

### 33.9.9 TIPCServerMsgQueue.Count

**Synopsis:** Number of messages in the queue

**Declaration:**
```pascal
Property Count : Integer
```

**Visibility:** public

**Access:** Read

**Description:** `Count` is the current number of messages in the queue.

**See also:** MaxCount (773)

### 33.9.10 TIPCServerMsgQueue.MaxCount

**Synopsis:** Maximum number of messages in the queue, 0 for unlimited

**Declaration:**
```pascal
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` (773) is equal to `MaxCount`, the `MaxAction` (773) property is examined to know what to do.

**See also:** Count (773), MaxAction (773)

### 33.9.11 TIPCServerMsgQueue.MaxAction

**Synopsis:** Action to take when the number of messages will exceed `MaxCount`.

**Declaration:**
```pascal
Property MaxAction : TIPCMessageOverflowAction
```

**Visibility:** public

**Access:** Read, Write

**Description:** `MaxAction` determines what will happen if the current `Count` (773) equals `MaxCount` (773) and a new message is put in the queue using `Push` (772):
Do nothing, just add the message
Discard the oldest message
Discard the new message

See also: TIPCServerMsgQueue.Count (773), TIPCServerMsgQueue.MaxCount (773), TIPCServerMsgQueue.Push (772), TIPCMessagesOverflowAction (763)

33.10 TSimpleIPC

33.10.1 Description
TSimpleIPC is the common ancestor for the TSimpleIPCServer (778) and TSimpleIPCCClient (775) classes. It implements some common properties between client and server.

See also: TSimpleIPCServer (778), TSimpleIPCCClient (775)

33.10.2 Property overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Properties</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>774</td>
<td>Active</td>
<td>rw</td>
<td>Communication channel active</td>
</tr>
<tr>
<td>774</td>
<td>ServerID</td>
<td>rw</td>
<td>Unique server identification</td>
</tr>
</tbody>
</table>

33.10.3 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 (774) identification.

See also: ServerID (774)

33.10.4 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 (774)
33.11 TSimpleIPCClient

33.11.1 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 (777) or SendStringMessage (777) calls.

See also: TSimpleIPCServer (778), TSimpleIPC (774), TIPCClientComm (764)

33.11.2 Method overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>776</td>
<td>Connect</td>
<td>Connect to the server</td>
</tr>
<tr>
<td>775</td>
<td>Create</td>
<td>Create a new instance of TSimpleIPCClient</td>
</tr>
<tr>
<td>775</td>
<td>Destroy</td>
<td>Remove the TSimpleIPCClient instance from memory</td>
</tr>
<tr>
<td>776</td>
<td>Disconnect</td>
<td>Disconnect from the server</td>
</tr>
<tr>
<td>777</td>
<td>SendMessage</td>
<td>Send a message to the server</td>
</tr>
<tr>
<td>777</td>
<td>SendStringMessage</td>
<td>Send a string message to the server</td>
</tr>
<tr>
<td>777</td>
<td>SendStringMessageFmt</td>
<td>Send a formatted string message</td>
</tr>
<tr>
<td>776</td>
<td>ServerRunning</td>
<td>Check if the server is running.</td>
</tr>
</tbody>
</table>

33.11.3 Property overview

<table>
<thead>
<tr>
<th>Properties</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ServerInstance</td>
<td>rw</td>
<td>Server instance identification</td>
</tr>
</tbody>
</table>

33.11.4 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 (775)

33.11.5 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.
33.11.6 TSimpleIPCClient.Connect

Synopsis: Connect to the server

Declaration: procedure Connect

Visibility: public

Description: Connect connects to the server indicated in the ServerID (774) and InstanceID (782) properties. Connect is called automatically if the Active (774) property is set to True.

After a successful call to Connect, messages can be sent to the server using SendMessage (777) or SendStringMessage (777).

Calling Connect if the connection is already open has no effect.

Errors: If creating the connection fails, an EIPCError (764) exception may be raised.

See also: ServerID (774), InstanceID (775), Active (774), SendMessage (777), SendStringMessage (777), Disconnect (776)

33.11.7 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 (776). Disconnect is called automatically if the Active (774) 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 (764) exception may be raised.

See also: Active (774), Connect (776)

33.11.8 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 (774) and InstanceID (782) 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 (776)
### 33.11.9 TSimpleIPCClient.SendMessage

**Synopsis:** Send a message to the server

**Declaration:**

```pascal
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 (764) exception is raised.

**See also:** Connect (776), SendStringMessage (777)

### 33.11.10 TSimpleIPCClient.SendStringMessage

**Synopsis:** Send a string message to the server

**Declaration:**

```pascal
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 (777) method.

**Errors:** Same as for SendMessage.

**See also:** SendMessage (777), Connect (776), SendStringMessageFmt (777)

### 33.11.11 TSimpleIPCClient.SendStringMessageFmt

**Synopsis:** Send a formatted string message

**Declaration:**

```pascal
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 (777) method.

**Errors:** Same as for SendMessage.

**See also:** SendMessage (777), Connect (776), SendStringMessage (777)

### 33.11.12 TSimpleIPCClient.ServerInstance

**Synopsis:** Server instance identification

**Declaration:**

```pascal
property ServerInstance : string
```

**Visibility:** public

**Access:** Read, Write
CHAPTER 33. REFERENCE FOR UNIT ‘SIMPLEIPC’

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 (782) property set to False, and should match the server’s InstanceID (782) property.

See also: ServerID (774), GLobal (782), InstanceID (782)

33.12 TSimpleIPCServer

33.12.1 Description

TSimpleIPCServer is the server side of the simple IPC communication protocol. The server program should create a TSimpleIPCServer instance, set its ServerID (774) property to a unique name for the system, and then set the Active (774) property to True (or call StartServer (779)).

After the server was started, it can check for availability of messages with the PeekMessage (780) call, and read the message with ReadMessage (780).

See also: TSimpleIPCClient (775), TSimpleIPC (774), TIPCServerComm (767)

33.12.2 Method overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>779</td>
<td>Create</td>
<td>Create a new instance of TSimpleIPCServer</td>
</tr>
<tr>
<td>779</td>
<td>Destroy</td>
<td>Remove the TSimpleIPCServer instance from memory</td>
</tr>
<tr>
<td>780</td>
<td>PeekMessage</td>
<td>Check if a client message is available.</td>
</tr>
<tr>
<td>780</td>
<td>ReadMessage</td>
<td>Read message from the queue</td>
</tr>
<tr>
<td>779</td>
<td>StartServer</td>
<td>Start the server</td>
</tr>
<tr>
<td>780</td>
<td>StopServer</td>
<td>Stop the server</td>
</tr>
</tbody>
</table>

33.12.3 Property overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Properties</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>782</td>
<td>Global</td>
<td>rw</td>
<td>Is the server reachable to all users or not</td>
</tr>
<tr>
<td>782</td>
<td>InstanceID</td>
<td>r</td>
<td>Instance ID</td>
</tr>
<tr>
<td>784</td>
<td>MaxAction</td>
<td>rw</td>
<td>Action to take when the number of messages will exceed MaxQueue.</td>
</tr>
<tr>
<td>784</td>
<td>MaxQueue</td>
<td>rw</td>
<td>Maximum number of messages in the queue, 0 for unlimited</td>
</tr>
<tr>
<td>781</td>
<td>Message</td>
<td>r</td>
<td>Last read message</td>
</tr>
<tr>
<td>781</td>
<td>MsgData</td>
<td>r</td>
<td>Last message data</td>
</tr>
<tr>
<td>781</td>
<td>MsgType</td>
<td>r</td>
<td>Last message type</td>
</tr>
<tr>
<td>783</td>
<td>OnMessage</td>
<td>rw</td>
<td>Event triggered when a message arrives</td>
</tr>
<tr>
<td>783</td>
<td>OnMessageError</td>
<td>rw</td>
<td>Event called when a new message has arrived, and the queue is full</td>
</tr>
<tr>
<td>783</td>
<td>OnMessageQueued</td>
<td>rw</td>
<td>Event called when a new message has arrived</td>
</tr>
<tr>
<td>783</td>
<td>OnThreadError</td>
<td>rw</td>
<td>Triggered when a thread reports an error</td>
</tr>
<tr>
<td>781</td>
<td>StringMessage</td>
<td>r</td>
<td>Last message as a string.</td>
</tr>
<tr>
<td>785</td>
<td>SynchronizeEvents</td>
<td>rw</td>
<td>Should events be run in the main thread?</td>
</tr>
<tr>
<td>784</td>
<td>Threaded</td>
<td>rw</td>
<td>Is the server running threaded or not?</td>
</tr>
<tr>
<td>782</td>
<td>ThreadError</td>
<td>r</td>
<td>Last thread error</td>
</tr>
<tr>
<td>782</td>
<td>ThreadExecuting</td>
<td>r</td>
<td>Is the message thread currently executing?</td>
</tr>
<tr>
<td>785</td>
<td>ThreadTimeout</td>
<td>rw</td>
<td>Timeout waiting for message</td>
</tr>
</tbody>
</table>
33.12.4 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 (779)

33.12.5 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 (779)

33.12.6 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 (767) 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 (784) 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 (774) property must be set.

Errors: If an error occurs a EIPCError (764) exception may be raised.

See also: TIPCServerComm (767), Active (774), ServerID (774), StopServer (780), TSimpleIPCServer.Threaded (784)
33.12.7 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 (767) descendent).

See also: TIPCServerComm (767), Active (774), ServerID (774), StartServer (779)

33.12.8 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 (780).

See also: ReadMessage (780)

33.12.9 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 (781) and TSimpleIPCServer.MsgData (781)

It is safe to call this even if a watch thread is started.

See also: TSimpleIPCServer.MsgType (781), TSimpleIPCServer.MsgData (781)

33.12.10 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 (781) 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 (780). It will also not clear the data buffer.

See also: StringMessage (781), MsgData (781), MsgType (781)
33.12.11 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 (780).

See also: GetMessageData (780)

33.12.12 TSimpleIPCServer.Message
Synopsis: Last read message

Declaration: Property Message : TIPCServerMsg

Visibility: public
Access: Read

Description: Message is the last read message (using TSimpleIPCServer.ReadMessage (780)) from the message queue.

See also: TSimpleIPCServer.ReadMessage (780)

33.12.13 TSimpleIPCServer.MsgType
Synopsis: Last message type

Declaration: Property MessageType : TMessageType

Visibility: public
Access: Read

Description: MessageType contains the message type of the last message.
This property will contain valid data only after a successful call to ReadMessage (780).

See also: ReadMessage (780)

33.12.14 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 (781) is better suited to read the data.
This property will contain valid data only after a successful call to ReadMessage (780).

See also: StringMessage (781), ReadMessage (780)
33.12.15 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 (774) property to form the unique server endpoint which a client should use.

See also: ServerID (774), Global (782)

33.12.16 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 (782), TSimpleIPCServer.StartServer (779), TSimpleIPCServer.OnThreadError (783), TSimpleIPCServer.Threaded (784)

33.12.17 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 (782), TSimpleIPCServer.StartServer (779), TSimpleIPCServer.OnThreadError (783), TSimpleIPCServer.Threaded (784)

33.12.18 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 (774), InstanceID (782)
33.12.19 TSimpleIPCServer.OnMessage

Synopsis: Event triggered when a message arrives

Declaration: `Property OnMessage : TNotifyEvent`

Visibility: published
Access: Read, Write

Description: `OnMessage` is called by `ReadMessage (780)` when a message has been read. The actual message data can be retrieved with one of the `StringMessage (781)`, `MsgData (781)` or `MsgType (781)` properties.

See also: `StringMessage (781)`, `MsgData (781)`, `MsgType (781)`

33.12.20 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 (780)`, `TSimpleIPCServer.OnMessageError (783)`

33.12.21 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, and `MaxAction (773)` is `ipcmoaError`.

33.12.22 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 (782)`

See also: `TSimpleIPCServer.ThreadExecuting (782)`, `TSimpleIPCServer.StartServer (779)`, `TSimpleIPCServer.OnThreadError (783)`, `TSimpleIPCServer.ThreadError (782)`, `TSimpleIPCServer.Threaded (784)`
33.12.23 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 (773) is equal to MaxQueue, the MaxAction (784) property is examined to know what to do.

See also: MaxAction (784)

33.12.24 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 (784) and a new message is put in the queue during PeekMessage (780):

- Do nothing, just add the message
- Discard the oldest message
- Discard the new message

See also: TSimpleIPCServer.MaxQueue (784), TSimpleIPCServer.PeekMessage (780), TIPCMessageOverflowAction (763)

33.12.25 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 (779). Trying to set it when the server is started will result in an error.

See also: StartServer (779)
### 33.12.26 TSimpleIPCServer.ThreadTimeout

**Synopsis:** Timeout waiting for message

**Declaration:**

```property
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 (779), TSimpleIPCServer.StopServer (780)

### 33.12.27 TSimpleIPCServer.SynchronizeEvents

**Synopsis:** Should events be run in the main thread?

**Declaration:**

```property
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 (784), StartServer (779)
Chapter 34

Reference for unit ’SQLDB’

34.1 Used units

Table 34.1: Used units by unit ’SQLDB’

<table>
<thead>
<tr>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>BufDataset</td>
<td>131</td>
</tr>
<tr>
<td>Classes</td>
<td>??</td>
</tr>
<tr>
<td>DB</td>
<td>319</td>
</tr>
<tr>
<td>sqlscript</td>
<td>??</td>
</tr>
<tr>
<td>SQLTypes</td>
<td>861</td>
</tr>
<tr>
<td>System</td>
<td>??</td>
</tr>
<tr>
<td>sysutils</td>
<td>??</td>
</tr>
</tbody>
</table>

34.2 Overview

The SQLDB unit defines four main classes to handle data in SQL based databases.

1. TSQLConnection (813) 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 (827) is a #fcl.db.TDataset (375) 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 (856) 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 (843) 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 (787).
34.3 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 (375) 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 (813) 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 (856) 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 (827) 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:

```
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';
  end;
end.
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 (827) can only execute 1 SQL statement during ExecSQL. If many SQL statements must be executed, TSQLScript (843) must be used.

There is much more to TSQLQuery than explained here: it can use parameters (see UsingParams (791)) and it can automatically update the data that you edit in it (see UpdateSQLs (790)).

See also: TSQLConnection (813), TSQLTransaction (856), TSQLQuery (827), TSQLConnector (824), TSQLScript (843), UsingParams (791), UpdateSQLs (790)

### 34.4 Using the universal TSQLConnector type

The normal procedure when using SQLDB is to use one of the TSQLConnection (813) 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 (824) 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 (825) 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 (800) 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 (824) 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 (813), TSQLConnector (824)

34.5 Retrieving Schema Information

Schema Information (lists of available database objects) can be retrieved using some specialized calls in TSQLConnection (813):

- TSQLConnection.GetTableNames (816) retrieves a list of available tables. The system tables can be requested.
- TSQLConnection.GetProcedureNames (816) retrieves a list of available stored procedures.
- TSQLConnection.GetFieldNames (816) 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 (807) 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 (816).
- **stSysTables** Retrieves the list of system Tables in database. This is used internally by TSQLConnection.GetTableNames (816) when the system tables are requested
- **stProcedures** Retrieves a list of stored procedures in database. This is used internally by TSQLConnection.GetProcedureNames (816).
- **stColumns** Retrieves the list of columns (fields) in a table. This is used internally by TSQLConnection.GetFieldNames (816).
- **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).

34.6 Automatic generation of update SQL statements

SQLDB (more in particular, TSQLQuery (827)) 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` (840) 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` (448) 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` (840) 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` (836), `InsertSQL` (836) and `DeleteSQL` (837) 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:

```sql
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 (791), TSQLQuery (827), UpdateSQL (836), InsertSQL (836), DeleteSQL (836)

### 34.7 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:

```sql
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:
CHAPTER 34. REFERENCE FOR UNIT 'SQLDB'

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.


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 (838), UpdateSQLs (790)

34.8 Constants, types and variables

34.8.1 Constants

DefaultMacroChar = '%'

DefaultMacroChar is the default macro delimiter to use in TSQLQuery (827)

DefaultSQLFormatSettings : TFormatSettings = (CurrencyFormat: 1; NegCurrFormat : 5; ThousandSeparator: #0; DecimalSeparator: '.'; CurrencyDecimals : 2; DateSeparator: '-'; TimeSeparator: ':'; ListSeparator: ' '; CurrencyString : '$'; ShortDateForma

DefaultSQLFormatSettings contains the default settings used when formatting date/time and other special values in Update SQL statements generated by the various TSQLConnection (813) descendents.

detActualSQL = sqltypes.detActualSQL

Alias for sqltypes.detActualSQL

detCommit = sqltypes.detCommit

Alias for sqltypes.detCommit

792
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 (821).

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 (861).

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
34.8.2 Types

\[\text{TCommitRollbackAction} = (\text{caNone}, \text{caCommit}, \text{caCommitRetaining}, \text{caRollback}, \text{caRollbackRetaining})\]

<table>
<thead>
<tr>
<th>Value</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>caCommit</td>
<td>Commit transaction</td>
</tr>
<tr>
<td>caCommitRetaining</td>
<td>Commit transaction, retaining transaction context</td>
</tr>
<tr>
<td>caNone</td>
<td>Do nothing</td>
</tr>
<tr>
<td>caRollback</td>
<td>Rollback transaction</td>
</tr>
<tr>
<td>caRollbackRetaining</td>
<td>Rollback transaction, retaining transaction context</td>
</tr>
</tbody>
</table>

\text{TCommitRollbackAction} is currently unused in SQLDB.

\text{TConnectionDefClass} = \text{Class of TConnectionDef}

\text{TConnectionDefClass} is used in the RegisterConnection (800) call to register a new TConnectionDef (802) instance.

\[\text{TConnInfoType} = (\text{citAll}, \text{citServerType}, \text{citServerVersion}, \text{citServerVersionString}, \text{citClientName}, \text{citClientVersion})\]

<table>
<thead>
<tr>
<th>Value</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>citAll</td>
<td>All connection information</td>
</tr>
<tr>
<td>citClientName</td>
<td>Client library name</td>
</tr>
<tr>
<td>citClientVersion</td>
<td>Client library version</td>
</tr>
<tr>
<td>citServerType</td>
<td>Server type description</td>
</tr>
<tr>
<td>citServerVersion</td>
<td>Server version as an integer number</td>
</tr>
<tr>
<td>citServerVersionString</td>
<td>Server version as a string</td>
</tr>
</tbody>
</table>

Connection information to be retrieved

\[\text{TConnOption} = (\text{sqSupportParams}, \text{sqSupportEmptyDatabaseName}, \text{sqEscapeSlash}, \text{sqEscapeRepeat}, \text{sqImplicitTransaction}, \text{sqLastInsertID}, \text{sqSupportReturning}, \text{sqSequences})\]
Table 34.4: Enumeration values for type TConnOption

<table>
<thead>
<tr>
<th>Value</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>sqEscapeRepeat</td>
<td>Escapes in string literals are done by repeating the character.</td>
</tr>
<tr>
<td>sqEscapeSlash</td>
<td>Escapes in string literals are done with backslash characters.</td>
</tr>
<tr>
<td>sqImplicitTransaction</td>
<td>Does the connection support implicit transaction management</td>
</tr>
<tr>
<td>sqLastInsertID</td>
<td>Does the connection support getting the ID for the last insert operation.</td>
</tr>
<tr>
<td>sqSequences</td>
<td>Are sequences supported.</td>
</tr>
<tr>
<td>sqSupportEmptyDatabaseName</td>
<td>Does the connection allow empty database names?</td>
</tr>
<tr>
<td>sqSupportParams</td>
<td>The connection type has native support for parameters.</td>
</tr>
<tr>
<td>sqSupportReturning</td>
<td>The connection type supports INSERT/UPDATE with RETURNING clause</td>
</tr>
</tbody>
</table>

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 (813) through the TSQLConnection.OnLog (821) event.

TDBEventTypes = sqltypes.TDBEventTypes

TDBEventTypes is a set of TDBEventType (797) values, which is used to filter the set of event messages that should be sent. The TSQLConnection.LogEvents (821) 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 (821) event handler and for the global GlobalDBLogHook (799) event handling hook. Sender will contain the TSQLConnection (813) 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 (797)
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 (807) 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 (825)

TSQLConnectionOption = (scoExplicitConnect, scoApplyUpdatesChecksRowsAffected)

Table 34.5: Enumeration values for type TSQLConnectionOption

<table>
<thead>
<tr>
<th>Value</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>scoApplyUpdatesChecksRowsAffected</td>
<td>ApplyUpdates will check that the RowsAffected is 1 after an update.</td>
</tr>
<tr>
<td>scoExplicitConnect</td>
<td>Require explicit connection to the database (default is implicit)</td>
</tr>
</tbody>
</table>

TSQLConnectionOption enumerates several options that can be set for TSQLConnection (813) instances using TSQLConnection.Options (822)

TSQLConnectionOptions = Set of TSQLConnectionOption

Set of TSQLConnectionOption

TSQLOption = (sqoKeepOpenOnCommit, sqoAutoApplyUpdates, sqoAutoCommit, sqoCancelUpdatesOnRefresh, sqoRefreshUsingSelect)

Table 34.6: Enumeration values for type TSQLOption

<table>
<thead>
<tr>
<th>Value</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>sqoAutoApplyUpdates</td>
<td>Call ApplyUpdates on Post or Delete</td>
</tr>
<tr>
<td>sqoAutoCommit</td>
<td>Call commit after every ApplyUpdates or ExecSQL</td>
</tr>
<tr>
<td>sqoCancelUpdatesOnRefresh</td>
<td>Cancel any pending updates when refresh is called</td>
</tr>
<tr>
<td>sqoKeepOpenOnCommit</td>
<td>Keep the dataset open after the query was committed (will fetch all records).</td>
</tr>
<tr>
<td>sqoRefreshUsingSelect</td>
<td>Force a refresh using the provided select instead of using RETURNING clause</td>
</tr>
</tbody>
</table>
TSQLQueryOption enumerates several options available to control the behaviour of an TSQL-Query (827) instance.

TSQLQueryOptions = Set of TSQLQueryOption

TSQLQueryOptions is the type of the TSQLQuery.Options (838) property.

TSQLSequenceApplyEvent = (saeOnNewRecord, saeOnPost)

<table>
<thead>
<tr>
<th>Value</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>saeOnNewRecord</td>
<td>Fetch an ID when a new record is appended to a dataset</td>
</tr>
<tr>
<td>saeOnPost</td>
<td>Fetch an ID when a new record is posted in the dataset</td>
</tr>
</tbody>
</table>

TSQLSequenceApplyEvent enumerates the moments when a new ID must be fetched for a sequence field.

TSQLTransactionOption = (stoUseImplicit, stoExplicitStart)

<table>
<thead>
<tr>
<th>Value</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>stoExplicitStart</td>
<td>Require explicit start of transactions by TSQLQuery</td>
</tr>
<tr>
<td>stoUseImplicit</td>
<td>Use implicit transaction control if the engine allows it.</td>
</tr>
</tbody>
</table>

TSQLTransactionOption enumerates several options that can be used to control the transaction behaviour of TSQLTransaction (856).

TSQLTransactionOptions = Set of TSQLTransactionOption

TSQLTransactionOptions is the property type of TSQLTransaction.Options (860).

TStatementType = sqltypes.TStatementType

TStatementType describes the kind of SQL statement that was entered in the SQL property of a TSQLQuery (827) component.

34.8.3 Variables

GlobalDBLogHook : TDBLogNotifyEvent

GlobalDBLogHook can be set in addition to local TSQLConnection.Onlog (821) 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.
CHAPTER 34. REFERENCE FOR UNIT ‘SQLDB’

34.9 Procedures and functions

34.9.1 GetConnectionDef

Synopsis: Search for a connection definition by name

Declaration: function GetConnectionDef(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 (800), TConnectionDef (802), TConnectionDef.TypeName (802)

34.9.2 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 (800). The names are the names as returned by TConnectionDef.TypeName (802)

See also: RegisterConnection (800), TConnectionDef.TypeName (802)

34.9.3 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 (802) descendant to register the connection type described in the TConnectionDef (802) descendant. The connection type is registered with the name as returned by TConnectionDef.TypeName (802).

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 (801).

Errors: if Def is Nil, access violations will occur.

See also: TConnectionDef (802), UnRegisterConnection (801)
CHAPTER 34. REFERENCE FOR UNIT ‘SQLDB’

34.9.4 UnRegisterConnection

Synopsis: Unregister a registered connection type

Declaration: procedure UnRegisterConnection(Def: TConnectionDefClass)
procedure UnRegisterConnection(ConnectionName: string)

Visibility: default

Description: UnRegisterConnection will unregister the connection Def. If a connection with ConnectionName or with name as returned by the TypeName (802) 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 (802), RegisterConnection (800)

34.10 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 (852) and TSQLQuery (827) 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.

34.11 ESQLDatabaseError

34.11.1 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 (786), ESQLDatabaseError.SQLState (??), ESQLDatabaseError.ErrorCode (??)

34.11.2 Method overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>801</td>
<td>CreateFmt</td>
<td>Create a new instance of ESQLDatabaseError</td>
</tr>
</tbody>
</table>

34.11.3 ESQLDatabaseError.CreateFmt

Synopsis: Create a new instance of ESQLDatabaseError
Declaration: constructor CreateFmt(const Fmt: string; const Args: Array of const;
    Comp: TComponent; AErrorCode: Integer;
    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 (??)

34.12 TConnectionDef

34.12.1 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 (800). A descendent class should override at least the TConnectionDef.TypeName (802) and TConnectionDef.ConnectionClass (803) methods to return the specific name and connection class to use.

See also: TConnectionDef.TypeName (802), TConnectionDef.ConnectionClass (803), RegisterConnection (800)

34.12.2 Method overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>804</td>
<td>ApplyParams</td>
<td>Apply parameters to an instance of TSQLConnection</td>
</tr>
<tr>
<td>803</td>
<td>ConnectionClass</td>
<td>Class to instantiate when this connection is requested</td>
</tr>
<tr>
<td>803</td>
<td>DefaultLibraryName</td>
<td>Default library name</td>
</tr>
<tr>
<td>803</td>
<td>Description</td>
<td>A descriptive text for this connection type</td>
</tr>
<tr>
<td>804</td>
<td>LoadedLibraryName</td>
<td>Currently loaded library.</td>
</tr>
<tr>
<td>803</td>
<td>LoadFunction</td>
<td>Return a function to call when the client library must be loaded</td>
</tr>
<tr>
<td>802</td>
<td>TypeName</td>
<td>Name of the connection type</td>
</tr>
<tr>
<td>804</td>
<td>UnLoadFunction</td>
<td>Return a function to call when the client library must be unloaded</td>
</tr>
</tbody>
</table>

34.12.3 TConnectionDef.TypeName

Synopsis: Name of the connection type

Declaration: class function TypeName : string; Virtual

Visibility: default

Description: TypeName is overridden by descendents to return the unique name for this connection type. It is what the TSQLConnection.ConnectorType (825) property should be set to select this connection type for the universal connection, and is the name that the GetConnectionDef (800) 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 (800)

This name can be an arbitrary name, no restrictions on the allowed characters exist.

See also: TSQLConnection.ConnectorType (825), GetConnectionDef (800), GetConnectionList (800), TConnectionDef.ConnectionClass (803)
34.12.4  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 (802)

34.12.5  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 (802)

34.12.6  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 (797), TConnectionDef (802), TLibraryUnLoadFunction (797)

34.12.7  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 (797), TConnectionDef.UnLoadFunction (804), TConnectionDef.DefaultLibraryName (803), TConnectionDef.LoadedLibraryName (804)
34.12.8 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 (797), TConnectionDef.LoadFunction (803), TConnectionDef.DefaultLibraryName (803), TConnectionDef.LoadedLibraryName (804)

34.12.9 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 (797), TLibraryUnLoadFunction (797), TConnectionDef.LoadFunction (803), TConnectionDef.UnLoadFunction (804), TConnectionDef.DefaultLibraryName (803)

34.12.10 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 (813) 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 (803). Params contains the contents of the TSQLConnection.Params (824) property of the connector.

See also: TSQLConnection.Params (824)

34.13 TCustomSQLQuery

34.13.1 Description

TCustomSQLQuery encapsulates a SQL statement: it implements all the necessary #fcl.db.TDataset (375) 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 (806) method.

Do not instantiate a TCustomSQLQuery class directly, instead use the TSQLQuery (827) descendent.
34.13.2 Method overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>808</td>
<td>ApplyUpdates</td>
<td>Apply updates and check result</td>
</tr>
<tr>
<td>805</td>
<td>Create</td>
<td>Create a new instance of TCustomSQLQuery.</td>
</tr>
<tr>
<td>808</td>
<td>Delete</td>
<td>Delete and optionally apply updates</td>
</tr>
<tr>
<td>805</td>
<td>Destroy</td>
<td>Destroy instance of TCustomSQLQuery</td>
</tr>
<tr>
<td>806</td>
<td>ExecSQL</td>
<td>Execute a SQL statement that does not return a result set</td>
</tr>
<tr>
<td>808</td>
<td>MacroByName</td>
<td>Convenience for Macros.ParamByName</td>
</tr>
<tr>
<td>807</td>
<td>ParamByName</td>
<td>Return parameter by name</td>
</tr>
<tr>
<td>808</td>
<td>Post</td>
<td>Post pending changes and optionally apply updates</td>
</tr>
<tr>
<td>806</td>
<td>Prepare</td>
<td>Prepare a query for execution</td>
</tr>
<tr>
<td>807</td>
<td>RowsAffected</td>
<td>Return the number of rows (records) affected by the last DML/DDL statement</td>
</tr>
<tr>
<td>807</td>
<td>SetSchemaInfo</td>
<td>SetSchemaInfo prepares the dataset to retrieve schema info.</td>
</tr>
<tr>
<td>806</td>
<td>UnPrepare</td>
<td>Unprepare a prepared query</td>
</tr>
</tbody>
</table>

34.13.3 Property overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Properties</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>809</td>
<td>Prepared</td>
<td>r</td>
<td>Is the query prepared?</td>
</tr>
<tr>
<td>809</td>
<td>SQLConnection</td>
<td>rw</td>
<td>Database as TSQLConnection</td>
</tr>
<tr>
<td>809</td>
<td>SQLTransaction</td>
<td>rw</td>
<td>Transaction as TSQLTransaction</td>
</tr>
</tbody>
</table>

34.13.4 TCustomSQLQuery.Create

**Synopsis:** Create a new instance of TCustomSQLQuery.

**Declaration:**
```pascal
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 (805)

34.13.5 TCustomSQLQuery.Destroy

**Synopsis:** Destroy instance of TCustomSQLQuery

**Declaration:**
```pascal
destructor Destroy; Override
```

**Visibility:** public

**Description:** Destroy cleans up the instance, closing the dataset and freeing all allocated resources.

See also: TCustomSQLQuery.Create (805)
### 34.13.6 TCustomSQLQuery.Prepare

**Synopsis:** Prepare a query for execution.

**Declaration:**
```plaintext
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 when ever 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 (830), TCustomSQLQuery.UnPrepare (806), TCustomSQLQuery.ExecSQL (806)

### 34.13.7 TCustomSQLQuery.UnPrepare

**Synopsis:** Unprepare a prepared query

**Declaration:**
```plaintext
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` (835) 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 (830), TCustomSQLQuery.Prepare (806), TCustomSQLQuery.ExecSQL (806)

### 34.13.8 TCustomSQLQuery.ExecSQL

**Synopsis:** Execute a SQL statement that does not return a result set

**Declaration:**
```plaintext
procedure ExecSQL; Virtual
```

**Visibility:** public

**Description:** `ExecSQL` will execute the statement in `TSQLQuery.SQL` (835), 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` (392) method must be used.

The SQL property should be a single SQL command. To execute multiple SQL statements, use the `TSQLScript` (843) component instead.

If the statement is a DML statement, the number of deleted/updated/inserted rows can be determined using `TCustomSQLQuery.RowsAffected` (807).

The `Database` and `Transaction` properties must be assigned before calling `ExecSQL`. Executing an empty SQL statement is also an error.
CHAPTER 34. REFERENCE FOR UNIT 'SQLDB'

Errors: If the server reports an error, an exception will be raised.

See also: TCustomSQLQuery.RowsAffected (807), TDataSet.Open (392)

34.13.9 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 (816) and TSQLConnection.GetProcedureNames (816) to get the necessary schema information from the database.

See also: TSQLConnection.GetTableNames (816), TSQLConnection.GetProcedureNames (816), RetrievingSchemaInformation (790)

34.13.10 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
(806).

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 (806), TSQLConnection (813)

34.13.11 TCustomSQLQuery.ParamByName

Synopsis: Return parameter by name

Declaration: function ParamByName(const AParamName: string) : TParam

Visibility: public

Description: ParamByName is a shortcut for Params.ParamByName (504). 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 (504), TSQLQuery.Params (838)

### 34.13.12 TCustomSQLQuery.MacroByName

**Synopsis:** Convenience for Macros.ParamByName

**Declaration:**

```pascal
function MacroByName(const AParamName: string) : TParam
```

**Visibility:** public

**Description:**

MacroByName checks Macros (804) 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 (504)

### 34.13.13 TCustomSQLQuery.ApplyUpdates

**Synopsis:** Apply updates and check result

**Declaration:**

```pascal
procedure ApplyUpdates(MaxErrors: Integer); Override; Overload
```

**Visibility:** public

**Description:**

ApplyUpdates is overridden in TCustomSQLQuery (804) to check the result of the update (using RowsAffected (807)).

See also: TSQLQueryOptions (799), TSQLQuery.Options (838)

### 34.13.14 TCustomSQLQuery.Post

**Synopsis:** Post pending changes and optionally apply updates

**Declaration:**

```pascal
procedure Post; Override
```

**Visibility:** public

**Description:**

Post is overridden from DB.TDataset.Post (786) to implement the auto-applyupdates mechanism: if TSQLQuery.Options (838) contains sqoAutoApplyUpdates, then ApplyUpdates is called as the last step of the Post operation.

See also: TSQLQuery.Options (838), TCustomSQLQuery.ApplyUpdates (808)

### 34.13.15 TCustomSQLQuery.Delete

**Synopsis:** Delete and optionally apply updates

**Declaration:**

```pascal
procedure Delete; Override
```

**Visibility:** public

**Description:**

Delete is overridden from DB.TDataset.Delete (786) to implement the auto-applyupdates mechanism: if TSQLQuery.Options (838) contains sqoAutoApplyUpdates, then ApplyUpdates is called as the last step of the Post operation.

See also: TSQLQuery.Options (838), TCustomSQLQuery.ApplyUpdates (808)
34.13.16 TCustomSQLQuery.\texttt{Prepared}

Synopsis: Is the query prepared?

Declaration: Property \texttt{Prepared} : Boolean

Visibility: public

Access: Read

Description: \texttt{Prepared} is true if \texttt{Prepare} (806) was called for this query, and an UnPrepare (806) was not done after that (take care: several actions call UnPrepare implicitly). Initially, \texttt{Prepared} will be False. Calling \texttt{Prepare} if the query was already prepared has no effect.

See also: TCustomSQLQuery.\texttt{Prepare} (806), TCustomSQLQuery.\texttt{UnPrepare} (806)

34.13.17 TCustomSQLQuery.\texttt{SQLConnection}

Synopsis: Database as TSQLConnection

Declaration: Property \texttt{SQLConnection} : TSQLConnection

Visibility: public

Access: Read, Write

Description: \texttt{SQLConnection} equals the \texttt{Database} property, but typecasted as a TSQLConnection (813) descendent.

See also: TSQLConnection (813), TCustomSQLQuery.\texttt{SQLTransaction} (809)

34.13.18 TCustomSQLQuery.\texttt{SQLTransaction}

Synopsis: Transaction as TSQLTransaction

Declaration: Property \texttt{SQLTransaction} : TSQLTransaction

Visibility: public

Access: Read, Write

Description: \texttt{SQLTransaction} equals the \texttt{Transaction} property, but typecasted as a TSQLTransaction (856) descendent.

See also: TSQLConnection (813), TCustomSQLQuery.\texttt{SQLConnection} (809)

34.14 TCustomSQLStatement

34.14.1 Description

\texttt{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 (786) component has. It can be used to execute SQL statements on a database that update data, execute stored procedures and DDL statements etc.

The \texttt{TCustomSQLStatement} is equivalent to TSQLQuery (827) in that it supports transactions (in the \texttt{Transaction} (820) property) and parameters (in the \texttt{Params} (824) property) and as such is a more versatile tool than executing queries using TSQLConnection.\texttt{ExecuteDirect} (815).
To use a TCustomSQLStatement is simple and similar to the use of TSQLQuery (827): set the Database (853) property to an existing connection component, and set the Transaction (855) property. After setting the SQL (855) property and filling Params (854), the SQL statement can be executed with the Execute (811) method.

TCustomSQLStatement is a parent class. Many of the properties are only made public (or published) in the TSQLStatement (852) class, which should be instantiated instead of the TCustomSQLStatement class.

See also: TSQLStatement (852), TDataSet (786), TSQLQuery (827), TSQLStatement.Transaction (855), TSQLStatement.Params (854), TCustomSQLStatement.Execute (811), TSQLStatement.Database (853), TSQLConnection.ExecuteDirect (815)

### 34.14.2 Method overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>810</td>
<td>Create</td>
<td>Create a new instance of TCustomSQLStatement</td>
</tr>
<tr>
<td>810</td>
<td>Destroy</td>
<td>Destroy a TCustomSQLStatement instance.</td>
</tr>
<tr>
<td>811</td>
<td>Execute</td>
<td>Execute the SQL statement.</td>
</tr>
<tr>
<td>812</td>
<td>ParamByName</td>
<td>Find a parameter by name</td>
</tr>
<tr>
<td>811</td>
<td>Prepare</td>
<td>Prepare the statement for execution</td>
</tr>
<tr>
<td>812</td>
<td>RowsAffected</td>
<td>Number of rows affected by the SQL statement.</td>
</tr>
<tr>
<td>811</td>
<td>Unprepare</td>
<td>Unprepare a previously prepared statement</td>
</tr>
</tbody>
</table>

### 34.14.3 Property overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Properties</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>812</td>
<td>Prepared</td>
<td>r</td>
<td>Is the statement prepared or not</td>
</tr>
</tbody>
</table>

### 34.14.4 TCustomSQLStatement.Create

**Synopsis:** Create a new instance of TCustomSQLStatement

**Declaration:**

```pascal
constructor Create(AOwner: TComponent); Override
```

**Visibility:** public

**Description:** Create initializes a new instance of TCustomSQLStatement and sets the SQL (855)Params (854), ParamCheck (853) and ParseSQL (855) to their initial values.

See also: TSQLStatement.SQL (855), TSQLStatement.Params (854), TSQLStatement.ParamCheck (853), TSQLStatement.ParseSQL (855), TSQLStatement.Destroy (852)

### 34.14.5 TCustomSQLStatement.Destroy

**Synopsis:** Destroy a TCustomSQLStatement instance.

**Declaration:**

```pascal
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 (853), TSQLStatement.Transaction (855)
34.14.6 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 (811) 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 (855) statement set or the Database (853) or Transaction (855) properties are empty.

See also: TSQLStatement.SQL (855), TSQLStatement.Database (853), TSQLStatement.Transaction (855), TCustomSQLStatement.Execute (811)

34.14.7 TCustomSQLStatement.Execute

Synopsis: Execute the SQL statement.

Declaration: procedure Execute

Visibility: public

Description: Execute executes the SQL (855) 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 (855) statement set or the Database (853) or Transaction (855) 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 (855), TSQLStatement.Database (853), TSQLStatement.Transaction (855)

34.14.8 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 (855), TSQLStatement.Database (853), TSQLStatement.Transaction (855), TCustomSQLStatement.Prepare (811)
34.14.9  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 (854) property.

Errors: If no parameter with the given name is found, an exception is raised.

See also: TSQLStatement.Params (854), TParams.ParamByName (786)

34.14.10  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 (811) was called. Not all databases may support this.

See also: TCustomSQLStatement.Execute (811)

34.14.11  TCustomSQLStatement.Prepared

Synopsis: Is the statement prepared or not

Declaration: Property Prepared : Boolean

Visibility: public

Access: Read

Description: Prepared equals True if Prepare (811) was called (implicitly or explicitly), it returns False if not. It can be set to True or False to call Prepare (811) or UnPrepare (811), respectively.

See also: TCustomSQLStatement.Prepare (811), TCustomSQLStatement.UnPrepare (811)

34.15  TServerIndexDefs

34.15.1  Description

TServerIndexDefs is a simple descendent of TIndexDefs (473) that implements the necessary methods to update the list of definitions using the TSQLConnection (813). It should not be used directly.

See also: TSQLConnection (813)

34.15.2  Method overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>813</td>
<td>Create</td>
<td>Create a new instance of TServerIndexDefs</td>
</tr>
<tr>
<td>813</td>
<td>Update</td>
<td>Updates the list of indexes</td>
</tr>
</tbody>
</table>
34.15.3  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 (804) descendent.

Errors: An EDatabaseError exception will be raised if ADataset is not a TCustomSQLQuery (804) descendent.

34.15.4  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 (813) methods for this.

34.16  TSQLConnection

34.16.1  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 (827), TSQLTransaction (856)

34.16.2  Method overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>814</td>
<td>Create</td>
<td>Create a new instance of TSQLConnection</td>
</tr>
<tr>
<td>818</td>
<td>CreateDB</td>
<td>Create a new Database on the server</td>
</tr>
<tr>
<td>814</td>
<td>Destroy</td>
<td>Destroys the instance of the connection.</td>
</tr>
<tr>
<td>818</td>
<td>DropDB</td>
<td>Procedure to drop or remove a Database</td>
</tr>
<tr>
<td>815</td>
<td>EndTransaction</td>
<td>End the Transaction associated with this connection</td>
</tr>
<tr>
<td>815</td>
<td>ExecuteDirect</td>
<td>Execute a piece of SQL code directly, using a Transaction if specified</td>
</tr>
<tr>
<td>817</td>
<td>GetConnectionInfo</td>
<td>Return some information about the connection</td>
</tr>
<tr>
<td>816</td>
<td>GetFieldNames</td>
<td>Gets a list of the field names in the specified table</td>
</tr>
<tr>
<td>818</td>
<td>GetNextValue</td>
<td>Get next value for a sequence</td>
</tr>
<tr>
<td>816</td>
<td>GetObjectNames</td>
<td>Return a collection of object names for a given type of object</td>
</tr>
<tr>
<td>816</td>
<td>GetProcedureNames</td>
<td>Gets a list of Stored Procedures in the Database</td>
</tr>
<tr>
<td>817</td>
<td>GetSchemaNames</td>
<td>Get database schema names</td>
</tr>
<tr>
<td>817</td>
<td>GetSequenceNames</td>
<td>Return a list of sequence names</td>
</tr>
<tr>
<td>817</td>
<td>GetStatementInfo</td>
<td>Get statement information</td>
</tr>
<tr>
<td>816</td>
<td>GetTableNames</td>
<td>Get a list of the tables in the specified database</td>
</tr>
<tr>
<td>815</td>
<td>StartTransaction</td>
<td>Start the Transaction associated with this Connection</td>
</tr>
</tbody>
</table>
34.16.3 Property overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Properties</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>820</td>
<td>CharSet</td>
<td>rw</td>
<td>The character set to be used in this database</td>
</tr>
<tr>
<td>822</td>
<td>Connected</td>
<td></td>
<td>Is a connection to the server active or not</td>
</tr>
<tr>
<td>819</td>
<td>ConnOptions</td>
<td>r</td>
<td>The set of Connection options being used in the Connection</td>
</tr>
<tr>
<td>823</td>
<td>DatabaseName</td>
<td></td>
<td>The name of the database to which connection is required.</td>
</tr>
<tr>
<td>819</td>
<td>FieldNameQuoteChars</td>
<td>rw</td>
<td>Characters used to quote field names.</td>
</tr>
<tr>
<td>821</td>
<td>Handle</td>
<td>r</td>
<td>Low level handle used by the connection</td>
</tr>
<tr>
<td>821</td>
<td>HostName</td>
<td>rw</td>
<td>The name of the host computer where the database resides</td>
</tr>
<tr>
<td>823</td>
<td>KeepConnection</td>
<td></td>
<td>Attempt to keep the connection open once it is established.</td>
</tr>
<tr>
<td>821</td>
<td>LogEvents</td>
<td>rw</td>
<td>Filter for events to log</td>
</tr>
<tr>
<td>823</td>
<td>LoginPrompt</td>
<td></td>
<td>Should SQLDB prompt for user credentials when a connection is activated.</td>
</tr>
<tr>
<td>821</td>
<td>OnLog</td>
<td>rw</td>
<td>Event handler for logging events</td>
</tr>
<tr>
<td>824</td>
<td>OnLogin</td>
<td></td>
<td>Event handler for login process</td>
</tr>
<tr>
<td>822</td>
<td>Options</td>
<td>rw</td>
<td>Options to observe for this connection</td>
</tr>
<tr>
<td>824</td>
<td>Params</td>
<td></td>
<td>Extra connection parameters</td>
</tr>
<tr>
<td>819</td>
<td>Password</td>
<td>rw</td>
<td>Password used when authenticating on the database server</td>
</tr>
<tr>
<td>822</td>
<td>Role</td>
<td>rw</td>
<td>Role in which the user is connecting to the database</td>
</tr>
<tr>
<td>820</td>
<td>Transaction</td>
<td>rw</td>
<td>Default transaction to be used for this connection</td>
</tr>
<tr>
<td>820</td>
<td>UserName</td>
<td>rw</td>
<td>The username for authentication on the database server</td>
</tr>
</tbody>
</table>

34.16.4 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 (813). After calling the inherited constructor, it will initialize the FieldNameQuoteChars (819) property and some other fields for internal use.

See also: FieldNameQuoteChars (819)

34.16.5 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.
### 34.16.6 TSQLConnection.StartTransaction

**Synopsis:** Start the Transaction associated with this Connection

**Declaration:**

```plaintext
procedure StartTransaction; Override
```

**Visibility:** public

**Description:**

StartTransaction is a convenience method which starts the default transaction (Transaction (820)). It is equivalent to

```plaintext
Connection.Transaction.StartTransaction
```

**Errors:** If no transaction is assigned, an exception will be raised.

**See also:** EndTransaction (815)

### 34.16.7 TSQLConnection.EndTransaction

**Synopsis:** End the Transaction associated with this connection

**Declaration:**

```plaintext
procedure EndTransaction; Override
```

**Visibility:** public

**Description:**

EndTransaction is a convenience method which ends the default transaction (TSQLConnection.Transaction (820)). It is equivalent to

```plaintext
Connection.Transaction.EndTransaction
```

**Errors:** If no transaction is assigned, an exception will be raised.

**See also:** StartTransaction (815)

### 34.16.8 TSQLConnection.ExecuteDirect

**Synopsis:** Execute a piece of SQL code directly, using a Transaction if specified

**Declaration:**

```plaintext
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 (827) component and its ExecSQL (806) method.

**Errors:** If no transaction is assigned, and no transaction is passed, an exception will be raised.

**See also:** TSQLQuery (827), ExecSQL (806)
34.16.9  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: #fc1.sqltypes.TSqlObjectIdentifierList (864)

34.16.10  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 (816), TSQLConnection.GetFieldNames (816)

34.16.11  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 (816), TSQLConnection.GetFieldNames (816)

34.16.12  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 (816), TSQLConnection.GetProcedureNames (816)
34.16.13  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 (816), TSQLConnection.GetProcedureNames (816), TSQLConnection.GetFieldNames (816)

34.16.14  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.

34.16.15  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 (796)

34.16.16  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 (801)
34.16.17 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 (823) property, the user credentials are taken from the TSQLConnection.UserName (820) and TSQLConnection.Password (819) 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 (818)

34.16.18 TSQLConnection.DropDB

Synopsis: Procedure to drop or remove a Database

Declaration: procedure DropDB; Virtual

Visibility: public

Description: DropDB does the opposite of CreateDB (818). 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 (818)

34.16.19 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 (817)
34.16.20 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 (796)

34.16.21 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.

34.16.22 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 (813) descendent to the default for that particular connection type.
See also: TSQLConnection (813)

34.16.23 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 (820) 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 (820), TSQLConnection.HostName (821)
34.16.24 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 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.

See also: TSQLTransaction

34.16.25 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, TSQLConnection.HostName, TSQLConnection.Role, TSQLConnection.CharSet

34.16.26 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, TSQLConnection.HostName, TSQLConnection.UserName, TSQLConnection.Role
34.16.27 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 (822), TSQLConnection.Password (819), TSQLConnection.UserName (820), TSQLConnection.DatabaseName (823), TSQLConnection.Charset (820)

34.16.28 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 (821) 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 (799) event handler.

See also: GlobalDBLogHook (799), TSQLConnection.LogEvents (821)

34.16.29 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 (821) and GlobalDBLogHook (799) event handlers. Only event types that are listed in this property will be sent.

See also: GlobalDBLogHook (799), TSQLConnection.OnLog (821)
34.16.30 TSQLConnection.Options

Synopsis: Options to observe for this connection.

Declaration: Property Options : TSQLConnectionOptions

Visibility: published

Access: Read, Write

Description: Options can be used to control the behaviour of SQLDB for this connection. The following options can be set:

- `scoExplicitConnect`: When set, the connection must be explicitly made. Default behaviour is for TSQLQuery to implicitly open the connection as needed.
- `scoApplyUpdatesChecksRowsAffected`: When set, whenever an update SQL Statement is executed during ApplyOptions of a dataset, the RowsAffected (812) is checked and must be equal to 1.

See also: TCustomSQLQuery.ApplyUpdates (808), TCustomSQLStatement.RowsAffected (812), TCustomSQLQuery.RowsAffected (807)

34.16.31 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 (819), TSQLConnection.UserName (820), TSQLConnection.DatabaseName (823), TSQLConnection.Hostname (821)

34.16.32 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 (823) on host HostName (821) using the credentials specified in UserName (820) and Password (819). If the connection or authentication fails, an exception is raised. This has the same effect as calling Open (362).

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 (786)
34.16.33 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 (819), TSQLConnection.UserName (820), TSQLConnection.Charset (820), TSQLConnection.Hostname (821)

34.16.34 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.

34.16.35 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 (824) event handler. The UserName (820) and Password (819) properties are ignored in this case.

See also: TSQLConnection.Password (819), TSQLConnection.UserName (820), OnLogin (824)
### 34.16.36 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 (819), TSQLConnection.UserName (820), TSQLConnection.Hostname (821), TSQLConnection.DatabaseName (823)

### 34.16.37 TSQLConnection.OnLogin

**Synopsis:** Event handler for login process

**Declaration:**

```
Property OnLogin :
```

**Visibility:** published

**Access:**

**Description:** `OnLogin` will be used when loginPrompt (823) 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 (823)

### 34.17 TSQLConnector

#### 34.17.1 Description

`TSQLConnector` implements a general connection type. When switching database backends, the normal procedure is to replace one instance of TSQLConnection (813) descendent with another, and connect all instances of TSQLQuery (827) and TSQLTransaction (856) to the new connection.

Using `TSQLConnector` avoids this: the type of connection can be set using the `ConnectorType` (825) property, which is a string property. The `TSQLConnector` class will (in the background) create the correct TSQLConnection (813) 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 (825), UniversalConnectors (789)

#### 34.17.2 Property overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Properties</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>825</td>
<td>ConnectorType</td>
<td>rw</td>
<td>Name of the connection type to use</td>
</tr>
</tbody>
</table>
34.17.3 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 (800) 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 (800)

34.18 TSQLCursor

34.18.1 Description

TSQLCursor is an abstract internal object representing a result set returned by a single SQL select statement (TSQLHandle (827)). It is used by the TSQLQuery (827) component to handle result sets returned by SQL statements.

This object must not be used directly.

See also: TSQLQuery (827), TSQLHandle (827)

34.19 TSQLDBFieldDef

34.19.1 Description

TSQLDBFieldDef is a SQLDB specific db.TFieldDef (786) descendent which has room for storing engine-specific data for the result set fields.

See also: TSQLDBFieldDef.SQLDBData (825)

34.19.2 Property overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Properties</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>825</td>
<td>SQLDBData</td>
<td>rw</td>
<td>Pointer to store engine-specific data for the result field</td>
</tr>
</tbody>
</table>

34.19.3 TSQLDBFieldDef.SQLDBData

Synopsis: Pointer to store engine-specific data for the result field

Declaration: Property SQLDBData : Pointer

Visibility: public

Access: Read, Write
Description: \texttt{SQLDBData} can be used by the \texttt{TSQLConnection} (813) descendents to store additional data about fields in a result set. It is not used by \texttt{TSQLQuery} itself.

See also: \texttt{TSQLQuery} (827)

### 34.20 TSQLDBFieldDefs

#### 34.20.1 Description

\texttt{TSQLDBFieldDefs} is a \texttt{TFieldDefs} descendent which creates \texttt{TSQLDBFieldDef} (825) descendents when a new field is added to the field set, to provide storage for engine-specific field data.

See also: \texttt{TSQLDBFieldDef} (825)

### 34.21 TSQLDBParam

#### 34.21.1 Description

\texttt{TSQLDBParam} is used to be able to create parameters which can store info about a field on which the parameter is based in the \texttt{TSQLDBParam.FieldDef} (826) field. This is useful when constructing update or insert queries. It can also store private data needed for the various \texttt{TSQLConnection} (813) descendents in the \texttt{TSQLDBParam.SQLDBData} (826) property.

See also: \texttt{TSQLDBFieldDef} (825), \texttt{TSQLDBParam.FieldDef} (826), \texttt{TSQLDBParam.SQLDBData} (826)

#### 34.21.2 Property overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Properties</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>826</td>
<td>FieldDef</td>
<td>rw</td>
<td>Field definition for update SQL</td>
</tr>
<tr>
<td>826</td>
<td>SQLDBData</td>
<td>rw</td>
<td>Private data for TSQLDB descendents.</td>
</tr>
</tbody>
</table>

### 34.21.3 TSQLDBParam.FieldDef

Synopsis: Field definition for update SQL

Declaration: \texttt{Property FieldDef : TFieldDef}

Visibility: public

Access: Read,Write

Description: \texttt{FieldDef} is used during generation of update SQL statements to store information about the field to be updated.

See also: \texttt{TSQLDBFieldDef} (825), \texttt{TSQLDBParam.SQLDBData} (826)

### 34.21.4 TSQLDBParam.SQLDBData

Synopsis: Private data for TSQLDB descendents.

Declaration: \texttt{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 (813) descendents.

See also: TSQLConnection (813), TSQLDBParam.FieldDef (826)

34.22 TSQLDBParams

34.22.1 Description

TSQLDBParams is a TParams descendent which creates TSQLDBParams (827) descendents when a new field is added to the field set, to provide storage for engine-specific field data.

See also: TSQLDBParam (826)

34.23 TSQLHandle

34.23.1 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 (827) component to communicate with the TSQLConnection (813) descendent.

This object must not be used directly.

See also: TSQLQuery (827), TSQLCursor (825)

34.24 TSQLQuery

34.24.1 Description

TSQLQuery exposes the properties and some methods introduced in TCustomSQLQuery (804). It encapsulates a single SQL statement: it implements all the necessary #fcl.db.TDataset (375) 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 (806) method.

Typically, the TSQLQuery.Database (835) property must be set once, the TSQLQuery.Transaction (835) property as well. Then the TSQLQuery.SQL (835) property can be set. Depending on the kind of SQL statement, the Open (392) 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 (856), TSQLConnection (813), TCustomSQLQuery.ExecSQL (806), TSQLQuery.SQL (835)
## 34.24.2 Property overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Properties</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>831</td>
<td>Active</td>
<td></td>
<td>Event triggered after refresh</td>
</tr>
<tr>
<td>831</td>
<td>AfterCancel</td>
<td></td>
<td></td>
</tr>
<tr>
<td>831</td>
<td>AfterClose</td>
<td></td>
<td></td>
</tr>
<tr>
<td>831</td>
<td>AfterDelete</td>
<td></td>
<td></td>
</tr>
<tr>
<td>832</td>
<td>AfterEdit</td>
<td></td>
<td></td>
</tr>
<tr>
<td>832</td>
<td>AfterInsert</td>
<td></td>
<td></td>
</tr>
<tr>
<td>832</td>
<td>AfterOpen</td>
<td></td>
<td></td>
</tr>
<tr>
<td>832</td>
<td>AfterPost</td>
<td></td>
<td></td>
</tr>
<tr>
<td>832</td>
<td>AfterRefresh</td>
<td></td>
<td>Event triggered after refresh</td>
</tr>
<tr>
<td>831</td>
<td>AutoCalcFields</td>
<td></td>
<td></td>
</tr>
<tr>
<td>832</td>
<td>BeforeCancel</td>
<td></td>
<td></td>
</tr>
<tr>
<td>833</td>
<td>BeforeClose</td>
<td></td>
<td></td>
</tr>
<tr>
<td>833</td>
<td>BeforeDelete</td>
<td></td>
<td></td>
</tr>
<tr>
<td>833</td>
<td>BeforeEdit</td>
<td></td>
<td></td>
</tr>
<tr>
<td>833</td>
<td>BeforeInsert</td>
<td></td>
<td></td>
</tr>
<tr>
<td>833</td>
<td>BeforeOpen</td>
<td></td>
<td></td>
</tr>
<tr>
<td>833</td>
<td>BeforePost</td>
<td></td>
<td></td>
</tr>
<tr>
<td>833</td>
<td>BeforeRefresh</td>
<td></td>
<td>Event triggered before refresh</td>
</tr>
<tr>
<td>834</td>
<td>BeforeScroll</td>
<td></td>
<td></td>
</tr>
<tr>
<td>835</td>
<td>Database</td>
<td></td>
<td>The <code>TSQLConnection</code> instance on which to execute SQL Statements</td>
</tr>
<tr>
<td>841</td>
<td>DataSource</td>
<td></td>
<td>Source for parameter values for unbound parameters</td>
</tr>
<tr>
<td>837</td>
<td>DeleteSQL</td>
<td></td>
<td>Statement to be used when deleting a new row in the database</td>
</tr>
<tr>
<td>830</td>
<td>FieldDefs</td>
<td></td>
<td>List of field definitions.</td>
</tr>
<tr>
<td>831</td>
<td>Filter</td>
<td></td>
<td></td>
</tr>
<tr>
<td>831</td>
<td>Filtered</td>
<td></td>
<td></td>
</tr>
<tr>
<td>837</td>
<td>IndexDefs</td>
<td></td>
<td>List of local index Definitions</td>
</tr>
<tr>
<td>836</td>
<td>InsertSQL</td>
<td></td>
<td>Statement to be used when inserting a new row in the database</td>
</tr>
<tr>
<td>839</td>
<td>MacroChar</td>
<td></td>
<td>Macro delimiter character</td>
</tr>
<tr>
<td>839</td>
<td>MacroCheck</td>
<td></td>
<td>Check for macros in the SQL statement.</td>
</tr>
<tr>
<td>839</td>
<td>Macros</td>
<td></td>
<td>Set of macros for this SQL statement.</td>
</tr>
<tr>
<td>830</td>
<td>MaxIndexesCount</td>
<td></td>
<td>Maximum allowed number of indexes.</td>
</tr>
<tr>
<td>834</td>
<td>OnCalcFields</td>
<td></td>
<td></td>
</tr>
<tr>
<td>834</td>
<td>OnDeleteError</td>
<td></td>
<td></td>
</tr>
<tr>
<td>834</td>
<td>OnEditError</td>
<td></td>
<td></td>
</tr>
<tr>
<td>834</td>
<td>OnFilterRecord</td>
<td></td>
<td></td>
</tr>
<tr>
<td>834</td>
<td>OnNewRecord</td>
<td></td>
<td></td>
</tr>
<tr>
<td>834</td>
<td>OnPostError</td>
<td></td>
<td></td>
</tr>
<tr>
<td>838</td>
<td>Options</td>
<td></td>
<td>Options controlling the behaviour of the dataset</td>
</tr>
<tr>
<td>839</td>
<td>ParamCheck</td>
<td></td>
<td>Should the SQL statement be checked for parameters</td>
</tr>
<tr>
<td>838</td>
<td>Params</td>
<td></td>
<td>Parameters detected in the SQL statement.</td>
</tr>
<tr>
<td>840</td>
<td>ParseSQL</td>
<td></td>
<td>Should the SQL statement be parsed or not</td>
</tr>
<tr>
<td>835</td>
<td>ReadOnly</td>
<td></td>
<td></td>
</tr>
<tr>
<td>837</td>
<td>RefreshSQL</td>
<td></td>
<td>Refresh query to re-fetch field values after a DB update</td>
</tr>
<tr>
<td>830</td>
<td>SchemaType</td>
<td></td>
<td>Schema type</td>
</tr>
<tr>
<td>841</td>
<td>Sequence</td>
<td></td>
<td>Sequence to use for auto-generating values using a sequence</td>
</tr>
<tr>
<td>842</td>
<td>ServerFilter</td>
<td></td>
<td>Append server-side filter to SQL statement</td>
</tr>
<tr>
<td>842</td>
<td>ServerFiltered</td>
<td></td>
<td>Should server-side filter be applied</td>
</tr>
<tr>
<td>842</td>
<td>ServerIndexDefs</td>
<td></td>
<td>List of indexes on the primary table of the query</td>
</tr>
<tr>
<td>835</td>
<td>SQL</td>
<td></td>
<td>The SQL statement to execute</td>
</tr>
<tr>
<td>830</td>
<td>StatementType</td>
<td></td>
<td>SQL statement type</td>
</tr>
<tr>
<td>835</td>
<td>Transaction</td>
<td></td>
<td>Transaction in which to execute SQL statements</td>
</tr>
<tr>
<td>840</td>
<td>UpdateMode</td>
<td></td>
<td>How to create update SQL statements.</td>
</tr>
<tr>
<td>836</td>
<td>UpdateSQL</td>
<td></td>
<td>Statement to be used when updating an existing row in the database</td>
</tr>
</tbody>
</table>
34.24.3 TSQLQuery.SchemaType

Synopsis: Schema type

Declaration: Property SchemaType :

Visibility: public

Access:

Description: SchemaType is the schema type set by TCustomSQLQuery.SetSchemaInfo (807). It determines what kind of schema information will be returned by the TSQLQuery instance.

See also: TCustomSQLQuery.SetSchemaInfo (807), RetrievingSchemaInformation (790)

34.24.4 TSQLQuery.StatementType

Synopsis: SQL statement type

Declaration: Property StatementType :

Visibility: public

Access:

Description: StatementType is determined during the Prepare (806) call when ParseSQL (840) is set to True. It gives an indication of the type of SQL statement that is being executed.

See also: TSQLQuery.SQL (835), TSQLQuery.ParseSQL (840), TSQLQuery.Params (838)

34.24.5 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.

34.24.6 TSQLQuery.FieldDefs

Synopsis: List of field definitions.

Declaration: Property FieldDefs :

Visibility: published

Access:
34.24.7 TSQLQuery.Active
Declaration: Property Active :
   Visibility: published
   Access:

34.24.8 TSQLQuery.AutoCalcFields
Declaration: Property AutoCalcFields :
   Visibility: published
   Access:

34.24.9 TSQLQuery.Filter
Declaration: Property Filter :
   Visibility: published
   Access:

34.24.10 TSQLQuery.Filtered
Declaration: Property Filtered :
   Visibility: published
   Access:

34.24.11 TSQLQuery.AfterCancel
Declaration: Property AfterCancel :
   Visibility: published
   Access:

34.24.12 TSQLQuery.AfterClose
Declaration: Property AfterClose :
   Visibility: published
   Access:

34.24.13 TSQLQuery.AfterDelete
Declaration: Property AfterDelete :
   Visibility: published
   Access:
### 34.24.14 TSQLQuery.AfterEdit

Declaration: Property AfterEdit :
Visibility: published
Access:

### 34.24.15 TSQLQuery.AfterInsert

Declaration: Property AfterInsert :
Visibility: published
Access:

### 34.24.16 TSQLQuery.AfterOpen

Declaration: Property AfterOpen :
Visibility: published
Access:

### 34.24.17 TSQLQuery.AfterPost

Declaration: Property AfterPost :
Visibility: published
Access:

### 34.24.18 TSQLQuery.AfterRefresh

Synopsis: Event triggered after refresh
Declaration: Property AfterRefresh :
Visibility: published
Access:

### 34.24.19 TSQLQuery.AfterScroll

Declaration: Property AfterScroll :
Visibility: published
Access:

### 34.24.20 TSQLQuery.BeforeCancel

Declaration: Property BeforeCancel :
Visibility: published
Access:
34.24.21 TSQLQuery.BeforeClose
Declaration: Property BeforeClose:
  Visibility: published
  Access:

34.24.22 TSQLQuery.BeforeDelete
Declaration: Property BeforeDelete:
  Visibility: published
  Access:

34.24.23 TSQLQuery.BeforeEdit
Declaration: Property BeforeEdit:
  Visibility: published
  Access:

34.24.24 TSQLQuery.BeforeInsert
Declaration: Property BeforeInsert:
  Visibility: published
  Access:

34.24.25 TSQLQuery.BeforeOpen
Declaration: Property BeforeOpen:
  Visibility: published
  Access:

34.24.26 TSQLQuery.BeforePost
Declaration: Property BeforePost:
  Visibility: published
  Access:

34.24.27 TSQLQuery.BeforeRefresh
Synopsis: Event triggered before refresh
Declaration: Property BeforeRefresh:
  Visibility: published
  Access:
34.24.28 TSQLQuery.BeforeScroll
Declaration: Property BeforeScroll :
Visibility: published
Access:

34.24.29 TSQLQuery.OnCalcFields
Declaration: Property OnCalcFields :
Visibility: published
Access:

34.24.30 TSQLQuery.OnDeleteError
Declaration: Property OnDeleteError :
Visibility: published
Access:

34.24.31 TSQLQuery.OnEditError
Declaration: Property OnEditError :
Visibility: published
Access:

34.24.32 TSQLQuery.OnFilterRecord
Declaration: Property OnFilterRecord :
Visibility: published
Access:

34.24.33 TSQLQuery.OnNewRecord
Declaration: Property OnNewRecord :
Visibility: published
Access:

34.24.34 TSQLQuery.OnPostError
Declaration: Property OnPostError :
Visibility: published
Access:
34.24.35 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 (813)) 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 (835) property must be set.

Multiple TSQLQuery instances can be connected to a database at the same time.

See also: TSQLQuery.Transaction (835), TSQLConnection (813), TSQLTransaction (856)

34.24.36 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 (856)) 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 (835), TSQLConnection (813), TSQLTransaction (856)

34.24.37 TSQLQuery.ReadOnly

Declaration: Property ReadOnly:

Visibility: published

Access:

34.24.38 TSQLQuery.SQL

Synopsis: The SQL statement to execute

Declaration: Property SQL:

Visibility: published

Access:
CHAPTER 34. REFERENCE FOR UNIT 'SQLDB'

Description: SQL is the SQL statement that will be executed when ExecSQL (806) is called, or Open (392) is called. It should contain a valid SQL statement for the connection to which the TSQLQuery (827) 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 (806)

If ParseSQL (840) is True, the SQL statement will be parsed and the Params (838) property will be updated with the names of the parameters found in the SQL statement.

See also: Using parameters

See also: TSQLQuery.ParseSQL (840), TSQLQuery.Params (838), TCustomSQLQuery.ExecSQL (806), TDataset.Open (392)

34.24.39 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 (835), TSQLQuery.UpdateSQL (836), TSQLQuery.DeleteSQL (837), TSQLQuery.UpdateMode (840), UsingParams (791), UpdateSQLS (790)

34.24.40 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 (835), TSQLQuery.InsertSQL (836), TSQLQuery.DeleteSQL (837), TSQLQuery.UpdateMode (840), UsingParams (791), UpdateSQLS (790)
34.24.41 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 (835), TSQLQuery.UpdateSQL (836), TSQLQuery.DeleteSQL (837), TSQLQuery.UpdateMode (840), UsingParams (791), UpdateSQLS (790)

34.24.42 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 (786) 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 (838)

See also: TField.Providerflags (786), TSQLQuery.Options (838)

34.24.43 TSQLQuery.IndexDefs
Synopsis: List of local index Definitions

Declaration: Property IndexDefs :

Visibility: published

Access:

Description: List of local index Definitions

See also: TCustomBufDataset.IndexDefs (786)
34.24.44 TSQLQuery.Options

Synopsis: Options controlling the behaviour of the dataset

Declaration: Property Options :

Visibility: published

Access:

Description: Options controls the behaviour of the dataset. The following options can be specified:

- **sqoKeepOpenOnCommit**: The default SQLDB behaviour is to close all datasets connected to a transaction when a transaction is committed or rolled back, which means that transactions must remain active as long as the dataset is open. This can create problems with locking of records etc. With this option set, the dataset will be kept open. Note that setting this option will cause SQLDB to fetch all records in the result set in memory.

- **sqoAutoApplyUpdates**: Setting this option will make TSQLQuery call ApplyUpdates after every Post or Delete operation.

- **sqoAutoCommit**: Setting this option will make TSQLQuery call commit after every ApplyUpdates

- **sqoCancelUpdatesOnRefresh**: Setting this option will cause TSQLQuery to abandon all pending changes when Refresh is called. The default behaviour is to raise an exception when Refresh is called and there are pending changes.

- **sqoPreferRefresh**: If the database engine supports RETURNING, then the returning mechanism is used to fetch field values after an update of the database. Setting this option will disable the use of RETURNING and will fetch updated or new values instead with the TSQLQuery.RefreshSQL (837) property or a constructed refresh SQL statement.

See also: TCustomSQLQuery.ApplyUpdates (808), TCustomSQLQuery.Post (808), TCustomSQLQuery.Delete (808)

34.24.45 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 (840) is True. For each named parameter in the SQL (835) property, a named item will appear in the collection, and the collection will be used to retrieve values from.

When Open (392) or ExecSQL (806) is called, and the Datasource (841) property is not Nil, then for each parameter for which no value was explicitly set (its Bound (498) 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 (835), TSQLQuery.ParseSQL (840), TParam.Bound (498), UsingParams (791), UpdateSQLS (790)
34.24.46 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 (488) instances should be added to the Params (838) 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 (488), Params (838), ParamCheck (839)

34.24.47 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 (838) 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 (838)

34.24.48 TSQLQuery.MacroCheck

Synopsis: Check for macros in the SQL statement.

Declaration: Property MacroCheck:

Visibility: published

Access:

Description: MacroCheck is the macro equivalent of ParamCheck (839): if set to True, it instructs the query component to check the SQL statement text for macros and add them to the Macros (839) collection.

See also: ParamCheck (839), Macros (839)

34.24.49 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.

### 34.24.50 TSQLQuery.ParseSQL

**Synopsis:** Should the SQL statement be parsed or not

**Declaration:** Property ParseSQL :

<table>
<thead>
<tr>
<th>Visibility</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>published</td>
<td></td>
<td>ParseSQL can be set to False to prevent TSQLQuery from parsing the SQL (835) property and attempting to detect the statement type or updating the Params (838) or StatementType (830) 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.</td>
</tr>
</tbody>
</table>

**See also:** TSQLQuery.SQL (835), TSQLQuery.Params (838)

### 34.24.51 TSQLQuery.UpdateMode

**Synopsis:** How to create update SQL statements.

**Declaration:** Property UpdateMode :

<table>
<thead>
<tr>
<th>Visibility</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>published</td>
<td></td>
<td>UpdateMode determines how the WHERE clause of the UpdateSQL (836) and DeleteSQL (837) statements are auto-generated.</td>
</tr>
</tbody>
</table>

| 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 (836), TSQLQuery.InsertSQL (836)

### 34.24.52 TSQLQuery.UsePrimaryKeyAsKey

**Synopsis:** Should primary key fields be marked pfInKey

**Declaration:** Property UsePrimaryKeyAsKey :

<table>
<thead>
<tr>
<th>Visibility</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>published</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
CHAPTER 34. REFERENCE FOR UNIT ‘SQLDB’

Description: UsePrimaryKeyAsKey can be set to True to let TSQLQuery fetch all server indexes and if there is a primary key, update the ProviderFlags (448) of the fields in the primary key with pfInKey (332).

The effect of this is that when UpdateMode (840) 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 (790).

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 (840), #fcl.bufdataset.TCustomBufDataset.Unidirectional (170), TField.ProviderFlags (448), pfInKey (332), UpdateSQLs (790)

34.24.53 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 (392) or ExecSQL (806) is called, and the Datasource property is not Nil then for each parameter for which no value was explicitly set (its Bound (498) 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 (838), ExecSQL (806), UsingParams (791), TParam.Bound (498)

34.24.54 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 (850) 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 (852)).

See also: TSQLSequence (850)
34.24.55  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 (835), when ServerFiltered (842) is set to True. If ServerFiltered (842) is set to False, ServerFilter is ignored.

If the dataset is active and ServerFiltered (842) is set to true, then changing this property will re-fetch the data from the server.

This property cannot be used when ParseSQL (840) 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 (840) is False will result in an exception.

See also: ServerFiltered (842)

34.24.56  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 (842). 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 (840) is False will result in an exception.

See also: ParseSQL (840), ServerFilter (842)

34.24.57  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 (840) is True. If a primary key is found, then the fields in it will be marked

See also: UsePrimaryKeyAsKey (840), Prepare (806)
34.25 TSQLScript

34.25.1 Description

TSQLScript is a component that can be used to execute many SQL statements using a TSQLQuery (827) component. The SQL statements are specified in a script TSQLScript.Script (847) separated by a terminator character (typically a semicolon (;)).

See also: TSQLTransaction (856), TSQLConnection (813), TCustomSQLQuery.ExecSQL (806), TSQLQuery.SQL (835)

34.25.2 Method overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>843</td>
<td>Create</td>
<td>Create a new TSQLScript instance.</td>
</tr>
<tr>
<td>844</td>
<td>Destroy</td>
<td>Remove the TSQLScript instance from memory.</td>
</tr>
<tr>
<td>844</td>
<td>Execute</td>
<td>Execute the script.</td>
</tr>
<tr>
<td>844</td>
<td>ExecuteScript</td>
<td>Convenience function, simply calls Execute</td>
</tr>
</tbody>
</table>

34.25.3 Property overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Properties</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>844</td>
<td>Aborted</td>
<td></td>
<td>True when the script was aborted</td>
</tr>
<tr>
<td>846</td>
<td>AutoCommit</td>
<td></td>
<td>Automatically commit every statement</td>
</tr>
<tr>
<td>848</td>
<td>CommentsinSQL</td>
<td></td>
<td>Should comments be passed to the SQL engine ?</td>
</tr>
<tr>
<td>845</td>
<td>DataBase</td>
<td>rw</td>
<td>Database on which to execute the script</td>
</tr>
<tr>
<td>847</td>
<td>Defines</td>
<td></td>
<td>Defined macros</td>
</tr>
<tr>
<td>847</td>
<td>Directives</td>
<td></td>
<td>List of directives</td>
</tr>
<tr>
<td>846</td>
<td>DollarStrings</td>
<td></td>
<td>List of alternate string delimiter token sequences</td>
</tr>
<tr>
<td>845</td>
<td>Line</td>
<td></td>
<td>Current line of execution in the script</td>
</tr>
<tr>
<td>845</td>
<td>OnDirectives</td>
<td>rw</td>
<td>Event handler if a directive is encountered</td>
</tr>
<tr>
<td>850</td>
<td>OnException</td>
<td></td>
<td>Exception handling event</td>
</tr>
<tr>
<td>847</td>
<td>Script</td>
<td></td>
<td>The script to execute</td>
</tr>
<tr>
<td>848</td>
<td>Terminator</td>
<td></td>
<td>Terminator character.</td>
</tr>
<tr>
<td>845</td>
<td>Transaction</td>
<td>rw</td>
<td>Transaction to use in the script</td>
</tr>
<tr>
<td>849</td>
<td>UseCommit</td>
<td></td>
<td>Control automatic handling of the COMMIT command.</td>
</tr>
<tr>
<td>849</td>
<td>UseDefines</td>
<td></td>
<td>Automatically handle pre-processor defines</td>
</tr>
<tr>
<td>846</td>
<td>UseDollarString</td>
<td></td>
<td>Enable support for dollarstrings</td>
</tr>
<tr>
<td>848</td>
<td>UseSetTerm</td>
<td></td>
<td>Should the SET TERM directive be recognized</td>
</tr>
</tbody>
</table>

34.25.4 TSQLScript.Create

Synopsis: Create a new TSQLScript instance.

Declaration: constructor Create(AOwner: TComponent); Override

Visibility: public

Description: Create instantiates a TSQLQuery (827) instance which will be used to execute the queries, and then calls the inherited constructor.

See also: TSQLScript.Destroy (844)
34.25.5 TSQLScript.Destroy
Synopsis: Remove the TSQLScript instance from memory.

Declaration: destructor Destroy; Override
Visibility: public

Description: Destroy frees the TSQLQuery (827) instance that was created during the Create constructor from memory and then calls the inherited destructor.

See also: TSQLScript.Create (843)

34.25.6 TSQLScript.Execute
Synopsis: Execute the script.

Declaration: procedure Execute; Override
Visibility: public

Description: Execute will execute the statements specified in Script (847) 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 (850) event handler is set, it may stop the event handler.

Errors: Handle errors using TSQLScript.OnException (850).

See also: Script (847), TSQLScript.OnException (850)

34.25.7 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.

34.25.8 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.
34.25.9 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.

34.25.10 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 (813) descendent. All SQL statements in the Script (847) property will be executed on this database.

See also: TSQLConnection (813), TSQLScript.Transaction (845), TSQLScript.Script (847)

34.25.11 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 (857) method.

See also: TSQLTransaction (856), TSQLTransaction.CommitRetaining (857), TSQLScript.Database (845)

34.25.12 TSQLScript.OnDirective

Synopsis: Event handler if a directive is encountered


  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 (847) 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 (847)
34.25.13 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 (845)

34.25.14 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 string appearing in the DollarStrings (846) 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 (846)

34.25.15 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 (846) 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 (846)
34.25.16 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 (845) event handler. The list should contain one word per line, no spaces allowed.
See also: OnDirective (845)

34.25.17 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 (849) 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 (849)

34.25.18 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 (848) property. Each of the statement will be executed on the database specified in Database (845). using the equivalent of the TCustomSQLQuery.ExecSQL (806) 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 (848) property. If that property is False, comments will be stripped prior to executing the SQL statements.
See also: TSQLScript.CommentsinSQL (848), TSQLScript.Terminator (848), TSQLScript.DataBase (845)
34.25.19 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 (845), Directives (847)

34.25.20 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 (847)

34.25.21 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 (848) 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 (848), TSQLScript.Script (847), TSQLScript.Directives (847)
### 34.25.22 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 (845) 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 (845), TSQLScript.Directives (847)

### 34.25.23 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
#undef
```

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 (847) 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 (847), TSQLScript.Defines (847)
34.25.24 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 (844)

34.26 TSQLSequence

34.26.1 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 (827) and its properties determine which field must be auto-generated, and at what moment this value must be generated.

See also: TSQLConnection.GetSequenceNames (817), TSQLConnection.GetNextValue (818)

34.26.2 Method overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>851</td>
<td>Apply</td>
<td>Apply a new value to a field</td>
</tr>
<tr>
<td>851</td>
<td>Assign</td>
<td>Assign one TSQLSequence to another</td>
</tr>
<tr>
<td>850</td>
<td>Create</td>
<td>Create a new instance</td>
</tr>
<tr>
<td>851</td>
<td>GetNextValue</td>
<td>Get a next value for the sequence</td>
</tr>
</tbody>
</table>

34.26.3 Property overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Properties</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>852</td>
<td>ApplyEvent</td>
<td>rw</td>
<td>When to apply the new value</td>
</tr>
<tr>
<td>851</td>
<td>FieldName</td>
<td>rw</td>
<td>Field to apply sequence to</td>
</tr>
<tr>
<td>852</td>
<td>IncrementBy</td>
<td>rw</td>
<td>Value to increment sequence with</td>
</tr>
<tr>
<td>852</td>
<td>SequenceName</td>
<td>rw</td>
<td>Sequence name to get values from</td>
</tr>
</tbody>
</table>

34.26.4 TSQLSequence.Create

Synopsis: Create a new instance

Declaration: constructor Create(AQuery: TCustomSQLQuery)

Visibility: public

Description: Create instantiates a new sequence. It requires a TSQLQuery (827) instance, which it needs to have access to a connection.

See also: TSQLQuery (827)
34.26.5 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 (851), TSQLSequence.SequenceName (852), TSQLSequence.IncrementBy (852)

34.26.6 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 (852) using TSQLSequence.GetNextValue (851) to the field TSQLSequence.FieldName (851) 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 (851), TSQLSequence.FieldName (851), TSQLSequence.SequenceName (852), TSQLSequence.IncrementBy (852)

34.26.7 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 (852) using TSQLSequence.IncrementBy (852)

Errors: If the dataset is not attached to a connected database, an exception will be raised.

See also: TSQLSequence.FieldName (851), TSQLSequence.SequenceName (852), TSQLSequence.IncrementBy (852), TSQLSequence.Apply (851), TSQLConnection.GetNextValue (818)

34.26.8 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 (851) is called. It must be a valid field name of the dataset that owns the TSQLSequence instance.

See also: Apply (851), SequenceName (852), IncrementBy (852)
34.26.9 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 (851) is called.

See also: TSQLSequence.Apply (851), TSQLSequence.GetNextValue (851), TSQLSequence.IncrementBy (852)

34.26.10 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 (852) when TSQLSequence.GetNextValue (851) is called.

See also: TSQLSequence.Apply (851), TSQLSequence.GetNextValue (851), TSQLSequence.SequenceName (852)

34.26.11 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 (389) or Append (380) is called) or when a newly inserted record is saved (when Post (392) is called).

See also: TSQLSequenceApplyEvent (799)

34.27 TSQLStatement

34.27.1 Description

TSQLStatement is a descendent of TCustomSQLStatement (809) which simply publishes the protected properties of that component.

See also: TCustomSQLStatement (809)
### 34.27.2 Property overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Properties</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>853</td>
<td>Database</td>
<td>Access</td>
<td>Database instance to execute statement on.</td>
</tr>
<tr>
<td>853</td>
<td>DataSource</td>
<td>Access</td>
<td>Datasource to copy parameter values from</td>
</tr>
<tr>
<td>854</td>
<td>MacroCheck</td>
<td>Access</td>
<td>Check for macros in the SQL statement.</td>
</tr>
<tr>
<td>854</td>
<td>Macros</td>
<td>Access</td>
<td>Set of macros for this SQL statement.</td>
</tr>
<tr>
<td>853</td>
<td>ParamCheck</td>
<td>Access</td>
<td>Should SQL be checked for parameters</td>
</tr>
<tr>
<td>854</td>
<td>Params</td>
<td>Access</td>
<td>List of parameters.</td>
</tr>
<tr>
<td>855</td>
<td>ParseSQL</td>
<td>Access</td>
<td>Parse the SQL statement</td>
</tr>
<tr>
<td>855</td>
<td>SQL</td>
<td>Access</td>
<td>The SQL statement to execute</td>
</tr>
<tr>
<td>855</td>
<td>Transaction</td>
<td>Access</td>
<td>The transaction in which the SQL statement should be executed.</td>
</tr>
</tbody>
</table>

#### 34.27.3 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 (813) descendent. It must be set, together with Transaction (855) in order to be able to call Prepare (811) or Execute (811).

See also: Transaction (855), Prepare (811), Execute (811)

#### 34.27.4 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 (412) instance. When Execute (811) 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 (412), Execute (811), #fcl.db.TParam.Bound (498)

#### 34.27.5 TSQLStatement.ParamCheck

**Synopsis:** Should SQL be checked for parameters

**Declaration:** Property ParamCheck:
- **Visibility:** published
- **Access:**
CHAPTER 34. REFERENCE FOR UNIT ‘SQLDB’

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 (488) instances should be added to the Params (854) 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 (488), TSQLStatement.Params (854), TSQLQuery.ParamCheck (839)

34.27.6 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 (855) statement (in the form :ParamName). The collection is filled automatically if the ParamCheck (853) property is True.

See also: SQL (855), ParamCheck (853), ParseSQL (855)

34.27.7 TSQLStatement.MacroCheck

Synopsis: Check for macros in the SQL statement.

Declaration: Property MacroCheck :

Visibility: published

Access:

Description: MacroCheck is the macro equivalent of ParamCheck (853): if set to True, it instructs the query component to check the SQL statement text for macros and add them to the Macros (854) collection.

See also: ParamCheck (853), Macros (854)

34.27.8 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 (854) 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 (854)
34.27.9 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 (855) 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 (855), ParamCheck (853)

34.27.10 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 (811) is called. If ParamCheck (853) 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 (854) 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. Therefore, no attempt is done to ensure that SQL contains a single SQL statement. However, error reporting and the RowsAffected (812) function may be wrong in such a case.

See also: ParseSQL (855), CheckParams (852), Params (854), Prepare (811), RowsAffected (812)

34.27.11 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 (853) property. This must be set before Prepare (811) is called.

See also: Database (853), Prepare (811), TSQLTransaction (856)
34.28 TSQLTransaction

34.28.1 Description

TSQLTransaction represents the transaction in which one or more TSQLQuery (827) 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 (813) used in an application.

See also: TSQLQuery (827), TSQLConnection (813)

34.28.2 Method overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>857</td>
<td>Commit</td>
<td>Commit the transaction, end transaction context.</td>
</tr>
<tr>
<td>857</td>
<td>CommitRetaining</td>
<td>Commit the transaction, retain transaction context.</td>
</tr>
<tr>
<td>856</td>
<td>Create</td>
<td>Create a new transaction</td>
</tr>
<tr>
<td>856</td>
<td>Destroy</td>
<td>Destroy transaction component</td>
</tr>
<tr>
<td>858</td>
<td>EndTransaction</td>
<td>End the transaction</td>
</tr>
<tr>
<td>857</td>
<td>Rollback</td>
<td>Roll back all changes made in the current transaction.</td>
</tr>
<tr>
<td>858</td>
<td>RollbackRetaining</td>
<td>Roll back changes made in the transaction, keep</td>
</tr>
<tr>
<td></td>
<td></td>
<td>transaction context</td>
</tr>
<tr>
<td>858</td>
<td>StartTransaction</td>
<td>Start a new transaction</td>
</tr>
</tbody>
</table>

34.28.3 Property overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Properties</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>859</td>
<td>Action</td>
<td>rw</td>
<td>Currently unused in SQLDB</td>
</tr>
<tr>
<td>859</td>
<td>Database</td>
<td>r</td>
<td>Database for which this component is handling connections</td>
</tr>
<tr>
<td>859</td>
<td>Handle</td>
<td>r</td>
<td>Low-level transaction handle</td>
</tr>
<tr>
<td>860</td>
<td>Options</td>
<td>rw</td>
<td>Transaction options</td>
</tr>
<tr>
<td>860</td>
<td>Params</td>
<td>rw</td>
<td>Transaction parameters</td>
</tr>
<tr>
<td>859</td>
<td>SQLConnection</td>
<td>rw</td>
<td>Database as TSQLConnection</td>
</tr>
</tbody>
</table>

34.28.4 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.

34.28.5 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.
34.28.6  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 (858) method. To commit data while retaining an active transaction, execute CommitRetaining (857) instead.

Errors: Executing Commit when no transaction is active will result in an exception. A transaction must be started by calling StartTransaction (858). If the database backend reports an error, an exception is raised as well.

See also: StartTransaction (858), CommitRetaining (857), Rollback (857), RollbackRetaining (858)

34.28.7  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 (857) instead.

Errors: Executing CommitRetaining when no transaction is active will result in an exception. A transaction must be started by calling StartTransaction (858). If the database backend reports an error, an exception is raised as well.

See also: StartTransaction (858), Retaining (857), Rollback (857), RollbackRetaining (858)

34.28.8  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 (858) 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 (858). If the database backend reports an error, an exception is raised as well.

See also: StartTransaction (858), CommitRetaining (857), Commit (857), RollbackRetaining (858)
34.28.9 \textbf{TSQLTransaction.RollbackRetaining}

\textbf{Synopsis:} Roll back changes made in the transaction, keep transaction context.

\textbf{Declaration:} procedure RollbackRetaining; Override

\textbf{Visibility:} public

\textbf{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 (857) instead.

\textbf{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).

\textbf{Errors:} Executing RollbackRetaining when no transaction is active will result in an exception. A transaction must be started by calling StartTransaction (858). If the database backend reports an error, an exception is raised as well.

\textbf{See also:} StartTransaction (858), Commit (857), Rollback (857), CommitRetaining (857)

34.28.10 \textbf{TSQLTransaction.StartTransaction}

\textbf{Synopsis:} Start a new transaction

\textbf{Declaration:} procedure StartTransaction; Override

\textbf{Visibility:} public

\textbf{Description:} StartTransaction starts a new transaction context. All changes written to the database must be confirmed with a Commit (857) or can be undone with a Rollback (857) call. Calling StartTransaction is equivalent to setting Active to True.

\textbf{Errors:} If StartTransaction is called while the transaction is still active, an exception will be raised.

\textbf{See also:} StartTransaction (858), Commit (857), Rollback (857), CommitRetaining (857), EndTransaction (858)

34.28.11 \textbf{TSQLTransaction.EndTransaction}

\textbf{Synopsis:} End the transaction

\textbf{Declaration:} procedure EndTransaction; Override

\textbf{Visibility:} public

\textbf{Description:} EndTransaction is equivalent to RollBack (857).

\textbf{See also:} RollBack (857)
**34.28.12 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 (813) descendent.

**34.28.13 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 (813)

**34.28.14 TSQLTransaction.Action**

Synopsis: Currently unused in SQLDB

Declaration: `Property Action : TCommitRollbackAction`

Visibility: published

Access: Read, Write

Description: `Action` is currently unused in SQLDB.

**34.28.15 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 (813) instance this transaction is handling transactions in. All datasets connected to this transaction component must have the same value for their `Database` (835) property.

See also: TSQLQuery.Database (835), TSQLConnection (813)
34.28.16 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 therefore depends on the type of connection.

See also: TSQLConnection (813)

34.28.17 TSQLTransaction.Options

Synopsis: Transaction options

Declaration: Property Options : TSQLTransactionOptions

Visibility: published
Access: Read, Write

Description: Options can be used to control the behaviour of SQLDB for this transaction.

- **stoUseImplicit** Use the implicit transaction support of the DB engine. This means that no explicit transaction start and stop commands will be sent to the server when the Commit or Rollback methods are called (effectively making them a no-op at the DB level).

- **stoExplicitStart** When set, whenever an SQL statement is executed, the transaction must have been started explicitly. Default behaviour is that the TSQLStatement (852) or TSQLQuery (827) start the transaction as needed.

See also: TSQLStatement (852), TSQLQuery (827)
Chapter 35

Reference for unit ’SQLTypes’

35.1 Used units

Table 35.1: Used units by unit ’SQLTypes’

<table>
<thead>
<tr>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Classes</td>
<td>??</td>
</tr>
<tr>
<td>System</td>
<td>??</td>
</tr>
<tr>
<td>sysutils</td>
<td>??</td>
</tr>
</tbody>
</table>

35.2 Constants, types and variables

35.2.1 Types

\[
\text{TDBEventTypes} = \{ \text{detCustom}, \text{detPrepare}, \text{detExecute}, \text{detFetch}, \text{detCommit}, \\
\text{detRollBack}, \text{detParamValue}, \text{detActualSQL} \}
\]

Table 35.2: Enumeration values for type TDBEventType

<table>
<thead>
<tr>
<th>Value</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>detActualSQL</td>
<td>Actual SQL as sent to engine message</td>
</tr>
<tr>
<td>detCommit</td>
<td>Transaction Commit message</td>
</tr>
<tr>
<td>detCustom</td>
<td>Custom event message</td>
</tr>
<tr>
<td>detExecute</td>
<td>SQL Execute message</td>
</tr>
<tr>
<td>detFetch</td>
<td>Fetch data message</td>
</tr>
<tr>
<td>detParamValue</td>
<td>Parameter name and value message</td>
</tr>
<tr>
<td>detPrepare</td>
<td>SQL prepare message</td>
</tr>
<tr>
<td>detRollBack</td>
<td>Transaction rollback message</td>
</tr>
</tbody>
</table>

\text{TDBEventTypes} describes the type of a database event message as generated by TSQLConnection (813) through the OnLog (821) event event.
TDBEventTypes is a set of TDBEventType (861) values, which is used to filter the set of event messages that should be sent. The TSQLConnection.LogEvents (821) 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 35.3: Enumeration values for type TSchemaType

<table>
<thead>
<tr>
<th>Value</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>stColumns</td>
<td>Columns in a table</td>
</tr>
<tr>
<td>stIndexes</td>
<td>Indexes for a table</td>
</tr>
<tr>
<td>stNoSchema</td>
<td>No schema</td>
</tr>
<tr>
<td>stPackages</td>
<td>Packages (for databases that support them)</td>
</tr>
<tr>
<td>stProcedureParams</td>
<td>Parameters for a stored procedure</td>
</tr>
<tr>
<td>stProcedures</td>
<td>Stored procedures in database</td>
</tr>
<tr>
<td>stSchemata</td>
<td>List of schemas in database(s) (for databases that support them)</td>
</tr>
<tr>
<td>stSequences</td>
<td>Sequences (for databases that support them)</td>
</tr>
<tr>
<td>stSysTables</td>
<td>System tables in database</td>
</tr>
<tr>
<td>stTables</td>
<td>User Tables in database</td>
</tr>
</tbody>
</table>

TSchemaType describes which schema information to retrieve in the TCustomSQLQuery.SetSchemaInfo (807) 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 35.4: Enumeration values for type TStatementType

<table>
<thead>
<tr>
<th>Value</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>stCommit</td>
<td>The statement commits a transaction</td>
</tr>
<tr>
<td>stDDL</td>
<td>The statement is a SQL DDL (Data Definition Language) statement</td>
</tr>
<tr>
<td>stDelete</td>
<td>The statement is a SQL DELETE statement</td>
</tr>
<tr>
<td>stExecProcedure</td>
<td>The statement executes a stored procedure</td>
</tr>
<tr>
<td>stGetSegment</td>
<td>The statement is a SQL get segment statement</td>
</tr>
<tr>
<td>stInsert</td>
<td>The statement is a SQL INSERT statement</td>
</tr>
<tr>
<td>stPutSegment</td>
<td>The statement is a SQL put segment statement</td>
</tr>
<tr>
<td>stRollback</td>
<td>The statement rolls back a transaction</td>
</tr>
<tr>
<td>stSelect</td>
<td>The statement is a SQL SELECT statement</td>
</tr>
<tr>
<td>stSelectForUpd</td>
<td>The statement selects data for update</td>
</tr>
<tr>
<td>stStartTrans</td>
<td>The statement starts a transaction</td>
</tr>
<tr>
<td>stUnknown</td>
<td>The statement type could not be detected.</td>
</tr>
<tr>
<td>stUpdate</td>
<td>The statement is a SQL UPDATE statement</td>
</tr>
</tbody>
</table>

TStatementType describes the kind of SQL statement that was entered in the SQL property of a TSQLQuery (827) component.

35.3 TSqlObjectIdentifier

35.3.1 Method overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>863</td>
<td>Create</td>
<td></td>
</tr>
<tr>
<td>863</td>
<td>FullName</td>
<td></td>
</tr>
</tbody>
</table>

35.3.2 Property overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Properties</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>864</td>
<td>ObjectName</td>
<td>rw</td>
<td></td>
</tr>
<tr>
<td>863</td>
<td>SchemaName</td>
<td>rw</td>
<td></td>
</tr>
</tbody>
</table>

35.3.3 TSqlObjectIdentifier.Create

Declaration: constructor Create (ACollection: TSqlObjectIdentifierList; const AObjectName: string; const ASchemaName: string)

Visibility: public

35.3.4 TSqlObjectIdentifier.FullName

Declaration: function FullName : string

Visibility: public

35.3.5 TSqlObjectIdentifier.SchemaName

Declaration: Property SchemaName : string
Visibility: public
Access: Read, Write

### 35.3.6 TSqlObjectIdentifier.ObjectName

**Declaration:**

```
Property ObjectName : string
```

Visibility: public
Access: Read, Write

### 35.4 TSqlObjectIdentifierList

#### 35.4.1 Description

TSqlObjectIdentifierList is a list class holding a list of TSqlObjectIdentifier elements. It is used in the TSQLConnection.GetObjectNames to return the list of objects requested.

See also: TSqlObjectIdentifier, #fcl.sqldb.TSQLConnection.GetObjectNames

#### 35.4.2 Method overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>864</td>
<td>AddIdentifier</td>
<td>Add an identifier to the list</td>
</tr>
</tbody>
</table>

#### 35.4.3 Property overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Properties</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>865</td>
<td>Identifiers</td>
<td>rw</td>
<td>Indexed access to all identifiers in the list.</td>
</tr>
</tbody>
</table>

#### 35.4.4 TSqlObjectIdentifierList.AddIdentifier

**Synopsis:** Add an identifier to the list

**Declaration:**

```
function AddIdentifier : TSqlObjectIdentifier; Overload
function AddIdentifier(const AObjectName: string;
const ASchemaName: string) : TSqlObjectIdentifier; Overload
```

Visibility: public

**Description:**
AddIdentifier adds an identifier to the list with schema name ASchemaName and object name AObjectName. It returns the new identifier object. Both arguments are optional, their default value is the empty string.

No checking on duplicate entries is performed.

See also: TSqlObjectIdentifier

864
35.4.5 TSqlObjectIdentifierList.Identifiers

Synopsis: Indexed access to all identifiers in the list.

Declaration: Property Identifiers[Index: Integer]: TSqlObjectIdentifier; default

Visibility: public

Access: Read, Write

Description: Identifiers gives indexed access to all TSQLObjectIdentifier objects in the list. Valid indexes run from 0 to Count - 1. It is the default property of the list class.

See also: TSQLObjectIdentifierList.AddIdentifier (864)
Chapter 36

Reference for unit ’streamcoll’

36.1 Used units

<table>
<thead>
<tr>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Classes</td>
<td>??</td>
</tr>
<tr>
<td>System</td>
<td>??</td>
</tr>
<tr>
<td>sysutils</td>
<td>??</td>
</tr>
</tbody>
</table>

36.2 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 (869) 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.

36.3 Procedures and functions

36.3.1 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 (868) and returns the read value. The value cannot be read and written across systems that have different endian values.
See also: ColReadDateTime (867), ColWriteBoolean (868), ColReadString (868), ColReadInteger (867), ColReadFloat (867), ColReadCurrency (867)

36.3.2 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 (868) and returns the read value. The value cannot be read and written across systems that have different endian values.

See also: ColReadDateTime (867), ColReadBoolean (866), ColReadString (868), ColReadInteger (867), ColReadFloat (867), ColWriteCurrency (868)

36.3.3 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 (868) and returns the read value. The value cannot be read and written across systems that have different endian values.

See also: ColWriteDateTime (868), ColReadBoolean (866), ColReadString (868), ColReadInteger (867), ColReadFloat (867), ColReadCurrency (867)

36.3.4 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 (869) and returns the read value. The value cannot be read and written across systems that have different endian values.

See also: ColReadDateTime (867), ColReadBoolean (866), ColReadString (868), ColReadInteger (867), ColWriteFloat (869), ColReadCurrency (867)

36.3.5 ColReadInteger

Synopsis: Read a 32-bit integer from a stream.

Declaration: function ColReadInteger(S: TStream) : Integer

Visibility: default
CHAPTER 36. REFERENCE FOR UNIT ‘STREAMCOLL’

Description: ColReadInteger reads a 32-bit integer from the stream S as it was written by ColWriteInteger (869) and returns the read value. The value cannot be read and written across systems that have different endian values.

See also: ColReadDateTime (867), ColReadBoolean (866), ColReadString (868), ColWriteInteger (869), ColReadFloat (867), ColReadCurrency (867)

36.3.6 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 (869) and returns the read value. The value cannot be read and written across systems that have different endian values.

See also: ColReadDateTime (867), ColReadBoolean (866), ColReadString (869), ColReadInteger (867), ColReadFloat (867), ColReadCurrency (867)

36.3.7 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 (866), ColWriteString (869), ColWriteInteger (869), ColWriteCurrency (868), ColWriteDateTime (868), ColWriteFloat (869)

36.3.8 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 (868), ColWriteString (869), ColWriteInteger (869), ColWriteDateTime (868), ColWriteFloat (869), ColReadCurrency (867)

36.3.9 ColWriteDateTime

Synopsis: Write a TDateTime value to stream

Declaration: procedure ColWriteDateTime(S: TStream; AValue: TDateTime)

Visibility: default

Description: ColWriteDateTime writes the TDateTime AValue to the stream S.

See also: ColReadDateTime (867), ColWriteBoolean (868), ColWriteString (869), ColWriteInteger (869), ColWriteFloat (869), ColWriteCurrency (868)

868
36.3.10 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 (868), ColWriteBoolean (868), ColWriteString (869), ColWriteInteger (869), ColReadFloat (867), ColWriteCurrency (868)

36.3.11 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 (868), ColWriteString (869), ColWriteInteger (867), ColWriteCurrency (868), ColWriteDateTime (868)

36.3.12 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 (868), ColReadString (868), ColWriteInteger (869), ColWriteCurrency (868), ColWriteDateTime (868), ColWriteFloat (869)

36.4 EStreamColl

36.4.1 Description
Exception raised when an error occurs when streaming the collection.

36.5 TStreamCollection

36.5.1 Description
TStreamCollection is a TCollection (??) descendent which implements 2 calls LoadFromStream (870) and SaveToStream (870) which load and save the contents of the collection to a stream. The collection items must be descendents of the TStreamCollectionItem (871) class for the streaming to work correctly.

Note that the stream must be used to load collections of the same type.

See also: TStreamCollectionItem (871)
36.5.2 Method overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>870</td>
<td>LoadFromStream</td>
<td>Load the collection from a stream</td>
</tr>
<tr>
<td>870</td>
<td>SaveToStream</td>
<td>Load the collection from the stream.</td>
</tr>
</tbody>
</table>

36.5.3 Property overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Properties</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>870</td>
<td>Streaming</td>
<td>r</td>
<td>Indicates whether the collection is currently being written to stream</td>
</tr>
</tbody>
</table>

36.5.4 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 (870). 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 (870)

36.5.5 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 (870). 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 (870)

36.5.6 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 (870), TStreamCollection.SaveToStream (870)
36.6 TStreamCollectionItem

36.6.1 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 (869) to save or load the item from the stream.

See also: TStreamCollection (869)
Chapter 37

Reference for unit ’streamex’

37.1 Used units

Table 37.1: Used units by unit ’streamex’

<table>
<thead>
<tr>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Classes</td>
<td>??</td>
</tr>
<tr>
<td>RtlConsts</td>
<td>??</td>
</tr>
<tr>
<td>System</td>
<td>??</td>
</tr>
<tr>
<td>sysutils</td>
<td>??</td>
</tr>
</tbody>
</table>

37.2 Overview

streamex implements some extensions to be used together with streams from the classes unit.

37.3 Constants, types and variables

37.3.1 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
37.4 TBidirBinaryObjectReader

37.4.1 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 (873), TDelphiReader (874)

37.4.2 Property overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Properties</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>873</td>
<td>Position</td>
<td>rw</td>
<td>Position in the stream</td>
</tr>
</tbody>
</table>

37.4.3 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 (874) class.

See also: TDelphiReader (874)

37.5 TBidirBinaryObjectWriter

37.5.1 Description

TBidirBinaryObjectWriter is a class descendent from TBinaryObjectWriter (??), which implements the necessary support for BiDi data.

See also: TBidirBinaryObjectWriter (873), TDelphiWriter (875)

37.5.2 Property overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Properties</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>873</td>
<td>Position</td>
<td>rw</td>
<td>Position in the stream</td>
</tr>
</tbody>
</table>

37.5.3 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 (875) class.

See also: TDelphiWriter (875)
37.6 TDelphiReader

37.6.1 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 (873) driver class.

See also: TDelphiWriter (875), TBidirBinaryObjectReader (873)

37.6.2 Method overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>874</td>
<td>GetDriver</td>
<td>Return the driver class as a TBidirBinaryObjectReader (873) class</td>
</tr>
<tr>
<td>874</td>
<td>Read</td>
<td>Read data from stream</td>
</tr>
<tr>
<td>874</td>
<td>ReadStr</td>
<td>Overrides the standard ReadStr method</td>
</tr>
</tbody>
</table>

37.6.3 Property overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Properties</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>875</td>
<td>Position</td>
<td>rw</td>
<td>Position in the stream</td>
</tr>
</tbody>
</table>

37.6.4 TDelphiReader.GetDriver

Synopsis: Return the driver class as a TBidirBinaryObjectReader (873) class

Declaration: function GetDriver : TBidirBinaryObjectReader

Visibility: public

Description: GetDriver simply returns the used driver and typecasts it as TBidirBinaryObjectReader (873) class.

See also: TBidirBinaryObjectReader (873)

37.6.5 TDelphiReader.ReadStr

Synopsis: Overrides the standard ReadStr method

Declaration: function ReadStr : string

Visibility: public

Description: ReadStr makes sure the TBidirBinaryObjectReader (873) methods are used, to store additional information about the stream position when reading the strings.

See also: TBidirBinaryObjectReader (873)

37.6.6 TDelphiReader.Read

Synopsis: Read data from stream

Declaration: procedure Read(var Buf; Count: LongInt); Override

Visibility: public
37.6.7 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 (874)

37.7 TDelphiWriter

37.7.1 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 (873) driver class.

See also: TDelphiReader (874), TBidirBinaryObjectWriter (873)

37.7.2 Method overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>876</td>
<td>FlushBuffer</td>
<td>Flushes the stream buffer</td>
</tr>
<tr>
<td>875</td>
<td>GetDriver</td>
<td>Return the driver class as a TBidirBinaryObjectWriter (873) class</td>
</tr>
<tr>
<td>876</td>
<td>Write</td>
<td>Write raw data to the stream</td>
</tr>
<tr>
<td>876</td>
<td>WriteStr</td>
<td>Write a string to the stream</td>
</tr>
<tr>
<td>876</td>
<td>WriteValue</td>
<td>Write value type</td>
</tr>
</tbody>
</table>

37.7.3 Property overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Properties</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>876</td>
<td>Position</td>
<td>rw</td>
<td>Position in the stream</td>
</tr>
</tbody>
</table>

37.7.4 TDelphiWriter.GetDriver

**Synopsis**: Return the driver class as a TBidirBinaryObjectWriter (873) class

**Declaration**: function GetDriver : TBidirBinaryObjectWriter

**Visibility**: public

**Description**: GetDriver simply returns the used driver and typecasts it as TBidirBinaryObjectWriter (873) class.

See also: TBidirBinaryObjectWriter (873)
37.7.5 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.

37.7.6 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.

37.7.7 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 (873) class methods, which update the position of the stream.

See also:TBidirBinaryObjectWriter (873)

37.7.8 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 (873) methods, which update the position of the stream.

See also:TBidirBinaryObjectWriter (873)

37.7.9 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 (873) instance used when streaming.

See also:TBidirBinaryObjectWriter (873)
37.8 TFileReader

37.8.1 Description
TFileReader is a TTextReader descendent that takes a file on disk as the source of text data.

See also: TStreamReader (883), TTextReader (887)

37.8.2 Method overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>877</td>
<td>Create</td>
<td>Create a new instance of TFileReader for a disk file</td>
</tr>
<tr>
<td>877</td>
<td>Destroy</td>
<td>Remove the TFileReader instance from memory.</td>
</tr>
<tr>
<td>878</td>
<td>ReadLine</td>
<td>Read a line of text</td>
</tr>
<tr>
<td>878</td>
<td>Reset</td>
<td>Reset the stream to its original position</td>
</tr>
</tbody>
</table>

37.8.3 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
```

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 (872) will be used. If the ABufferSize argument is less than MIN_BUFFER_SIZE (872), 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 (884), TFileStream (??), TFileStream.Create (??)

37.8.4 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 (877)
37.8.5 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 (888) method.

See also: TTextReader.Reset (888)

37.8.6 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 (888)

See also: TTextReader.Close (888)

37.8.7 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 (888) method.

See also: Eof (889), TTextReader.ReadLine (888)

37.9 TStreamHelper

37.9.1 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 (?)
### 37.9.2 Method overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>882</td>
<td>ReadDouble</td>
<td>Read a double-precision floating point value from the stream</td>
</tr>
<tr>
<td>881</td>
<td>ReadDWordBE</td>
<td>Read a DWord from the stream, big endian</td>
</tr>
<tr>
<td>879</td>
<td>ReadDWordLE</td>
<td>Read a DWord from the stream, little endian</td>
</tr>
<tr>
<td>881</td>
<td>ReadQWordBE</td>
<td>Read a QWord from the stream, big endian</td>
</tr>
<tr>
<td>879</td>
<td>ReadQWordLE</td>
<td>Read a QWord from the stream, little endian</td>
</tr>
<tr>
<td>882</td>
<td>ReadSingle</td>
<td>Read a single-precision floating point value from the stream</td>
</tr>
<tr>
<td>881</td>
<td>ReadWordBE</td>
<td>Read a Word from the stream, big endian</td>
</tr>
<tr>
<td>879</td>
<td>ReadWordLE</td>
<td>Read a Word from the stream, little endian</td>
</tr>
<tr>
<td>883</td>
<td>WriteDouble</td>
<td>Write a double-precision floating point value to the stream</td>
</tr>
<tr>
<td>882</td>
<td>WriteDWordBE</td>
<td>Write a DWord value, big endian</td>
</tr>
<tr>
<td>880</td>
<td>WriteDWordLE</td>
<td>Write a DWord value, little endian</td>
</tr>
<tr>
<td>882</td>
<td>WriteQWordBE</td>
<td>Write a QWord value, big endian</td>
</tr>
<tr>
<td>880</td>
<td>WriteQWordLE</td>
<td>Write a QWord value, little endian</td>
</tr>
<tr>
<td>883</td>
<td>WriteSingle</td>
<td>Write a single-precision floating point value to the stream</td>
</tr>
<tr>
<td>881</td>
<td>WriteWordBE</td>
<td>Write a word value, big endian</td>
</tr>
<tr>
<td>880</td>
<td>WriteWordLE</td>
<td>Write a word value, little endian</td>
</tr>
</tbody>
</table>

### 37.9.3 TStreamHelper.ReadWordLE

**Synopsis:** Read a Word from the stream, little endian

**Declaration:**

```delphi
definition 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 (879)`, `TStreamHelper.ReadQWordLE (879)`, `TStreamHelper.WriteWordLE (880)`

### 37.9.4 TStreamHelper.ReadDWordLE

**Synopsis:** Read a DWord from the stream, little endian

**Declaration:**

```delphi
definition 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 (879)`, `TStreamHelper.ReadQWordLE (879)`, `TStreamHelper.WriteDWordLE (880)`

### 37.9.5 TStreamHelper.ReadQWordLE

**Synopsis:** Read a QWord from the stream, little endian

**Declaration:**

```delphi
definition 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 (879)`, `TStreamHelper.ReadDWordLE (879)`, `TStreamHelper.WriteQWordLE (880)`

### 37.9.6 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 (879)`, `TStreamHelper.WriteDWordLE (880)`, `TStreamHelper.WriteQWordLE (880)`

### 37.9.7 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 (879)`, `TStreamHelper.WriteWordLE (880)`, `TStreamHelper.WriteQWordLE (880)`

### 37.9.8 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 (879)`, `TStreamHelper.WriteDWordLE (880)`, `TStreamHelper.WriteWordLE (880)`
37.9.9 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 (881), TStreamHelper.ReadQWordBE (881), TStreamHelper.WriteWordBE (881)

37.9.10 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 (881), TStreamHelper.ReadQWordBE (881), TStreamHelper.WriteDWordBE (882)

37.9.11 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 (881), TStreamHelper.ReadDWordBE (881), TStreamHelper.WriteQWordBE (882)

37.9.12 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 (881), TStreamHelper.WriteDWordBE (882), TStreamHelper.WriteQWordBE (882)
### 37.9.13 TStreamHelper.WriteDWordBE

**Synopsis:** Write a DWord value, big endian

**Declaration:**
```delphi
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 (881), TStreamHelper.WriteWordBE (881), TStreamHelper.WriteQWordBE (882)

### 37.9.14 TStreamHelper.WriteQWordBE

**Synopsis:** Write a QWord value, big endian

**Declaration:**
```delphi
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 (881), TStreamHelper.WriteDWordBE (882), TStreamHelper.WriteWordBE (881)

### 37.9.15 TStreamHelper.ReadSingle

**Synopsis:** Read a single-precision floating point value from the stream

**Declaration:**
```delphi
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 (882), TStreamHelper.WriteSingle (883)

### 37.9.16 TStreamHelper.ReadDouble

**Synopsis:** Read a double-precision floating point value from the stream

**Declaration:**
```delphi
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 (882), TStreamHelper.WriteDouble (883)
37.9.17 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 (882), TStreamHelper.WriteDouble (883)

37.9.18 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 (882), TStreamHelper.WriteSingle (883)

37.10 TStreamReader

37.10.1 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 (887)

37.10.2 Method overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>884</td>
<td>Close</td>
<td>Close and possibly free the stream</td>
</tr>
<tr>
<td>884</td>
<td>Create</td>
<td>Create a new instance of TStreamReader from a stream</td>
</tr>
<tr>
<td>884</td>
<td>Destroy</td>
<td>Destroy the TStreamReader instance.</td>
</tr>
<tr>
<td>885</td>
<td>ReadLine</td>
<td>Read a line of text</td>
</tr>
<tr>
<td>884</td>
<td>Reset</td>
<td>Reset the stream to its original position</td>
</tr>
</tbody>
</table>

37.10.3 Property overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Properties</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>885</td>
<td>BaseStream</td>
<td>r</td>
<td>The stream with the text data</td>
</tr>
<tr>
<td>885</td>
<td>OwnsStream</td>
<td>rw</td>
<td>Should the stream be freed on close</td>
</tr>
</tbody>
</table>
37.10.4 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 (872) will be used. If the ABufferSize argument is less than MIN_BUFFER_SIZE (872), 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 (884), TStream (??)

37.10.5 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 (TStreamReader.BaseStream (885)) if OwnsStream is True.

See also: TStreamReader.OwnsStream (885), TStreamReader.Create (884), TStreamReader.BaseStream (885)

37.10.6 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 (888) method.

Errors: If the source stream (TStreamReader.BaseStream (885)) is not seekable, then this method may raise an exception.
See also: TStreamReader.BaseStream (885)

37.10.7 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 (885), TStreamReader.OwnsStream (885)
37.10.8  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 (888) method.

See also: Eof (889), TTextReader.ReadLine (888)

37.10.9  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 (884) call.
Manipulating the stream between calls to TStreamReader.ReadLine (885) is not allowed, it will lead to wrong data being read from the stream.

See also: TStreamReader.Create (884), TStreamReader.OwnsStream (885)

37.10.10  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 (885) must be freed when TStreamReader.Close (884) is called or when the TStreamReader instance is destroyed.

See also: TStreamReader.BaseStream (885), TStreamReader.Close (884)

37.11  TStringReader

37.11.1  Description

TStringReader is a TTextReader descendent that takes a single string as the source of text data.

See also: TTextReader (887)
37.11.2 Method overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>887</td>
<td>Close</td>
<td>Close and possibly free the stream</td>
</tr>
<tr>
<td>886</td>
<td>Create</td>
<td>Create a new instance of TStringReader from a string</td>
</tr>
<tr>
<td>886</td>
<td>Destroy</td>
<td>Free the TStringReader instance.</td>
</tr>
<tr>
<td>887</td>
<td>ReadLine</td>
<td>Read a line of text</td>
</tr>
<tr>
<td>886</td>
<td>Reset</td>
<td>Reset the stream to its original position</td>
</tr>
</tbody>
</table>

37.11.3 TStringReader.Create

Synopsis: Create a new instance of TStringReader 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 (872) will be used. If the ABufferSize argument is less than MIN_BUFFER_SIZE (872), then MIN_BUFFER_SIZE bytes will be used.

Errors: If AStream is Nil, an rtl.sysutils.EArgumentException exception will be raised.

See also: TStringReader.Destroy (884), TStream (?)

37.11.4 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 (886)

37.11.5 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 (888) method.

See also: TStringReader.BaseStream (885)
37.11.6 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 (888) method.

See also: TTextReader.Close (888)

37.11.7 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 (888) method.

See also: Eof (889), TTextReader.ReadLine (888)

37.12 TTextReader

37.12.1 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 (888) procedure. Several descendents of this class exist which implement the reader interface for several sources of text data: TStreamReader (883), TFileReader (877), TStringReader (885).

See also: TStreamReader (883), TFileReader (877), TStringReader (885), ReadLine (888)

37.12.2 Method overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>888</td>
<td>Close</td>
<td>Close the text data stream</td>
</tr>
<tr>
<td>888</td>
<td>Create</td>
<td>Instantiate a new instance.</td>
</tr>
<tr>
<td>888</td>
<td>ReadLine</td>
<td>Read a line of text</td>
</tr>
<tr>
<td>888</td>
<td>Reset</td>
<td>Reset the reader to the start position</td>
</tr>
</tbody>
</table>

37.12.3 Property overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Properties</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>889</td>
<td>Eof</td>
<td>r</td>
<td>Check whether the end of the text data is returned.</td>
</tr>
</tbody>
</table>
37.12.4 TTextReader.Create
Synopsis: Instantiate a new instance.
Declaratio n: constructor Create; Virtual
Visibility: public
Description: Create does nothing in TTextReader.
See also: ReadLine (888)

37.12.5 TTextReader.Reset
Synopsis: Reset the reader to the start position
Declaratio n: 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 (888), TTextReader.ReadLine (888)

37.12.6 TTextReader.Close
Synopsis: Close the text data stream
Declaratio n: procedure Close; Virtual; Abstract
Visibility: public
Description: Close closes the data stream. No ReadLine (888) call can be performed after a call to Close.
See also: TTextReader.Reset (888), TTextReader.ReadLine (888)

37.12.7 TTextReader.ReadLine
Synopsis: Read a line of text
Declaratio n: 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 (889)
37.12.8 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 (888)

37.13 TWindowedStream

37.13.1 Description

TWindowedStream is a TStream (872) 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 (872), TWindowedStream.Create (889)

37.13.2 Method overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>889</td>
<td>Create</td>
<td>Initialize a new instance of TWindowedStream</td>
</tr>
<tr>
<td>890</td>
<td>Destroy</td>
<td>Destroy the TWindowedStream instance.</td>
</tr>
<tr>
<td>890</td>
<td>Read</td>
<td>Read data from the stream</td>
</tr>
<tr>
<td>890</td>
<td>Seek</td>
<td>Reposition the stream</td>
</tr>
<tr>
<td>890</td>
<td>Write</td>
<td>Read data to the stream</td>
</tr>
</tbody>
</table>

37.13.3 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.
37.13.4 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 (889)

37.13.5 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 (890), TWindowedStream.Seek (890)

37.13.6 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 (890), TWindowedStream.Seek (890)

37.13.7 TWindowedStream.Seek
Synopsis: Reposition the stream
Declaration: function Seek(const aOffset: Int64; aOrigin: TSeekorigin) : Int64
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 raised. If the combination of `AOrigin` and `aOffset` falls outside the valid window of the stream, an `EReadError` exception is raised.

See also: `TWindowedStream.Read (890)`, `TWindowedStream.Write (890)`
Chapter 38

Reference for unit ’StreamIO’

38.1 Used units

Table 38.1: Used units by unit ’StreamIO’

<table>
<thead>
<tr>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Classes</td>
<td>??</td>
</tr>
<tr>
<td>System</td>
<td>??</td>
</tr>
<tr>
<td>syutils</td>
<td>??</td>
</tr>
</tbody>
</table>

38.2 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.

38.3 Procedures and functions

38.3.1 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 (893)
38.3.2 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 (892).

Errors: An invalid class reference will be returned if the file was not associated with a stream.

See also: AssignStream (892), TStream (?)
Chapter 39

Reference for unit ’syncobjs’

39.1 Used units

Table 39.1: Used units by unit ’syncobjs’

<table>
<thead>
<tr>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>System</td>
<td>??</td>
</tr>
<tr>
<td>sysutils</td>
<td>??</td>
</tr>
</tbody>
</table>

39.2 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 (895) class is a wrapper around low-level critical section routines (semaphores or mutexes). The TEventObject (898) class can be used to send messages between threads (also known as conditional variables in POSIX threads).

39.3 Constants, types and variables

39.3.1 Constants

INFINITE = Cardinal(-1)

Constant denoting an infinite timeout.

39.3.2 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 (898) class.

TEventHandle = Pointer

TEventHandle is an opaque type and should not be used in user code.

TWaitResult = (wrSignaled, wrTimeout, wrAbandoned, wrError)

<table>
<thead>
<tr>
<th>Value</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>wrAbandoned</td>
<td>Wait operation was abandoned.</td>
</tr>
<tr>
<td>wrError</td>
<td>An error occurred during the wait operation.</td>
</tr>
<tr>
<td>wrSignaled</td>
<td>Event was signaled (triggered)</td>
</tr>
<tr>
<td>wrTimeout</td>
<td>Time-out period expired</td>
</tr>
</tbody>
</table>

TWaitResult is used to report the result of a wait operation.

### 39.4 ELockException

**39.4.1 Description**

ELockException is provided for Delphi compatibility. It is not used in FPC.

See also: ESyncObjectException (895), ELockRecursionException (895)

### 39.5 ELockRecursionException

**39.5.1 Description**

ELockRecursionException is provided for Delphi compatibility. It is not used in FPC.

See also: ESyncObjectException (895), ELockException (895)

### 39.6 ESyncoObjectException

**39.6.1 Description**

ESyncoObjectException is used in the constructor of TEventObject (898) to indicate failure to create a basic event.

See also: TEventObject (898), ELockRecursionException (895), ELockException (895)

### 39.7 TCriticalSection

**39.7.1 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):

```plaintext
// 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 (896), Release (896)

### 39.7.2 Method overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>896</td>
<td>Acquire</td>
<td>Enter the critical section</td>
</tr>
<tr>
<td>897</td>
<td>Create</td>
<td>Create a new critical section.</td>
</tr>
<tr>
<td>898</td>
<td>Destroy</td>
<td>Destroy the critical section instance</td>
</tr>
<tr>
<td>897</td>
<td>Enter</td>
<td>Alias for Acquire</td>
</tr>
<tr>
<td>897</td>
<td>Leave</td>
<td>Alias for Release</td>
</tr>
<tr>
<td>896</td>
<td>Release</td>
<td>Leave the critical section</td>
</tr>
<tr>
<td>897</td>
<td>TryEnter</td>
<td>Try and obtain the critical section</td>
</tr>
</tbody>
</table>

### 39.7.3 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 (896)

### 39.7.4 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 (896)
### 39.7.5 TCriticalSection.Enter

**Synopsis:** Alias for Acquire

**Declaration:**
```
procedure Enter
```

**Visibility:** public

**Description:** Enter just calls Acquire (896).

See also: Leave (897), Acquire (896)

### 39.7.6 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 (897)

### 39.7.7 TCriticalSection.Leave

**Synopsis:** Alias for Release

**Declaration:**
```
procedure Leave
```

**Visibility:** public

**Description:** Leave just calls Release (896)

See also: Release (896), Enter (897)

### 39.7.8 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 (898)
39.7.9 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 (897), Acquire (896)

39.8 TEventObject

39.8.1 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 (895)

39.8.2 Method overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>898</td>
<td>Create</td>
<td>Create a new event object</td>
</tr>
<tr>
<td>899</td>
<td>destroy</td>
<td>Clean up the event and release from memory</td>
</tr>
<tr>
<td>899</td>
<td>ResetEvent</td>
<td>Reset the event</td>
</tr>
<tr>
<td>899</td>
<td>SetEvent</td>
<td>Set the event</td>
</tr>
<tr>
<td>899</td>
<td>WaitFor</td>
<td>Wait for the event to be set.</td>
</tr>
</tbody>
</table>

39.8.3 Property overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Properties</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>900</td>
<td>ManualReset</td>
<td>r</td>
<td>Should the event be reset manually</td>
</tr>
</tbody>
</table>

39.8.4 TEventObject.Create

Synopsis: Create a new event object

Declaration: constructor Create(EventAttributes: PSecurityAttributes; AManualReset: Boolean; InitialState: Boolean; const Name: string)

Visibility: public
Description: Create creates a new event object with unique name AName. 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 (900), ResetEvent (899)

### 39.8.5 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 (898)

### 39.8.6 TEventObject.ResetEvent

Synopsis: Reset the event

Declaration: procedure ResetEvent

Visibility: public

Description: ResetEvent turns off the event. Any WaitFor (899) operation will suspend the calling thread.

See also: SetEvent (899), WaitFor (899)

### 39.8.7 TEventObject.SetEvent

Synopsis: Set the event

Declaration: procedure SetEvent

Visibility: public

Description: SetEvent sets the event. If the ManualReset (900) is True any thread that was waiting for the event to be set (using WaitFor (899)) 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 (899), ManualReset (900)

### 39.8.8 TEventObject.WaitFor

Synopsis: Wait for the event to be set.

Declaration: function WaitFor(Timeout: Cardinal) : TWaitResult

Visibility: public
CHAPTER 39. REFERENCE FOR UNIT 'SYNCOBJS'

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 (899)) 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 (899)

39.8.9 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 (899) must be called manually after a thread was notified.

See also: ResetEvent (899)

39.9 THandleObject

39.9.1 Description

THandleObject is an abstract parent class for synchronization classes that need to store an operating system handle. It introduces a property Handle (901) 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 (901)

39.9.2 Method overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>901</td>
<td>destroy</td>
<td>Free the instance</td>
</tr>
</tbody>
</table>

39.9.3 Property overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Properties</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>901</td>
<td>Handle</td>
<td>r</td>
<td>Handle for this object</td>
</tr>
<tr>
<td>901</td>
<td>LastError</td>
<td>r</td>
<td>Last operating system error</td>
</tr>
</tbody>
</table>
39.9.4 THandleObject.destroy
Synopsis: Free the instance
Declaration: destructor destroy; Override
Visibility: public
Description: Destroy does nothing in the Free Pascal implementation of THandleObject.

39.9.5 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.

39.9.6 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 (901).
See also: Handle (901)

39.10 TSimpleEvent

39.10.1 Description
TSimpleEvent is a simple descendent of the TEventObject (898) class. It creates an event with no name, which must be reset manually, and which is initially not set.
See also: TEventObject (898), TSimpleEvent.Create (902)

39.10.2 Method overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>902</td>
<td>Create</td>
<td>Creates a new TSimpleEvent instance</td>
</tr>
</tbody>
</table>
39.10.3 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 (898) with Nil for the security attributes, an empty name, AManualReset set to True, and InitialState to False.

See also: TEventObject.Create (898)

39.11 TSynchoObject

39.11.1 Description

TSynchoObject is an abstract synchronization resource object. It implements 2 virtual methods Acquire (902) which can be used to acquire the resource, and Release (902) to release the resource.

See also: Acquire (902), Release (902)

39.11.2 Method overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>902</td>
<td>Acquire</td>
<td>Acquire synchronization resource</td>
</tr>
<tr>
<td>902</td>
<td>Release</td>
<td>Release previously acquired synchronization resource</td>
</tr>
</tbody>
</table>

39.11.3 TSynchoObject.Acquire

Synopsis: Acquire synchronization resource

Declaration: procedure Acquire; Virtual

Visibility: default

Description: Acquire does nothing in TSynchoObject. Descendent classes must override this method to acquire the resource they manage.

See also: Release (902)

39.11.4 TSynchoObject.Release

Synopsis: Release previously acquired synchronization resource

Declaration: procedure Release; Virtual

Visibility: default

Description: Release does nothing in TSynchoObject. Descendent classes must override this method to release the resource they acquired through the Acquire (902) call.

See also: Acquire (902)
Chapter 40

Reference for unit 'URIParser'

40.1 Used units

Table 40.1: Used units by unit 'URIParser'

<table>
<thead>
<tr>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>System</td>
<td>??</td>
</tr>
</tbody>
</table>

40.2 Overview

The URIParser unit contains a basic type (TURI (905)) and some routines for the parsing (ParseURI (904)) and construction (EncodeURI (903)) 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..

40.3 Constants, types and variables

40.3.1 Types

40.4 Procedures and functions

40.4.1 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 (904)
### 40.4.2 FilenameToURI

**Synopsis:** Construct a URI from a filename

**Declaration:**

```pascal
definition 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 (905)

### 40.4.3 IsAbsoluteURI

**Synopsis:** Check whether a URI is absolute.

**Declaration:**

```pascal
definition 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 (904), URIToFileName (905)

### 40.4.4 ParseURI

**Synopsis:** Parse a URI and split it into its constituent parts

**Declaration:**

```pascal
definition 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:

```text
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 (903)

### 40.4.5 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

40.4.6 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 (904), FilenameToURI (904)

40.5 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 (904) call. It contains the contents of a 
URI, parsed out in its various pieces.
Chapter 41

Reference for unit ’Zipper’

41.1 Used units

<table>
<thead>
<tr>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>BaseUnix</td>
<td>??</td>
</tr>
<tr>
<td>Classes</td>
<td>??</td>
</tr>
<tr>
<td>System</td>
<td>??</td>
</tr>
<tr>
<td>sysutils</td>
<td>??</td>
</tr>
<tr>
<td>ZStream</td>
<td>944</td>
</tr>
</tbody>
</table>

41.2 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 Nomss Nzali. It is used in the FCL to implement the `TCompressionStream` class.

41.3 Constants, types and variables

41.3.1 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`
Chapter 41. Reference for Unit 'Zipper'

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

### 41.3.2 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 (912)

**FreeListArray** = Array[FIRSTENTRY..TABLESIZE] of Word

Helper type in decoding the zip file.

**FreeListPtr** = ^FreeListArray

Pointer to FreeListArray (910)

**TablePtr** = ^CodeArray

Pointer to CodeArray (910)

**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
CHAPTER 41. REFERENCE FOR UNIT 'ZIPPER'

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.

41.4 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;

 911
CHAPTER 41. REFERENCE FOR UNIT 'ZIPPER'

External_Attributes : LongWord;
Local_Header_Offset : LongWord;
end

This record contains the structure for a file header within the central directory.

41.5 CodeRec

CodeRec = packed record
  Child : SmallInt;
  Sibling : SmallInt;
  Suffix : Byte;
end

Small LZW compression helper type

41.6 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.

41.7 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.
41.8 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

41.9 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.

41.10 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;

913
This record appears at the end of the central directory

41.11 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

41.12 EZipError

41.12.1 Description

Exception raised for errors in TZipper and TUnZipper

41.13 TCompressor

41.13.1 Description

This object compresses a stream into a compressed zip stream.

41.13.2 Method overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>915</td>
<td>Compress</td>
<td>Compresses input stream to output stream</td>
</tr>
<tr>
<td>915</td>
<td>Create</td>
<td>Creates a TCompressor (914) object</td>
</tr>
<tr>
<td>915</td>
<td>Terminate</td>
<td>Halts the compressor by setting the Terminated property to True</td>
</tr>
<tr>
<td>915</td>
<td>ZipBitFlag</td>
<td>Current bit</td>
</tr>
<tr>
<td>915</td>
<td>ZipID</td>
<td>Identifier for type of compression</td>
</tr>
<tr>
<td>915</td>
<td>ZipVersionReqd</td>
<td>ZIP version required in the method</td>
</tr>
</tbody>
</table>

41.13.3 Property overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Properties</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>916</td>
<td>BufferSize</td>
<td>r</td>
<td>Size of the buffer used for compression</td>
</tr>
<tr>
<td>916</td>
<td>Cr32Val</td>
<td>rw</td>
<td>Running CRC32 value</td>
</tr>
<tr>
<td>916</td>
<td>OnPercent</td>
<td>rw</td>
<td>Threshold percentage which triggers an OnProgress update</td>
</tr>
<tr>
<td>916</td>
<td>OnProgress</td>
<td>rw</td>
<td>Event handler signalled to indicate the completion percentage for the compressor</td>
</tr>
<tr>
<td>916</td>
<td>Terminated</td>
<td>r</td>
<td>Set to True when the Terminate method is called</td>
</tr>
</tbody>
</table>
41.13.4 TCompressor.Create
Synopsis: Creates a TCompressor (914) object
Declaration: constructor Create(AInFile: TStream; AOutFile: TStream;
ABufSize: LongWord); Virtual
Visibility: public

41.13.5 TCompressor.Compress
Synopsis: Compresses input stream to output stream
Declaration: procedure Compress; Virtual; Abstract
Visibility: public

41.13.6 TCompressor.ZipID
Synopsis: Identifier for type of compression
Declaration: class function ZipID : Word; Virtual; Abstract
Visibility: public

41.13.7 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.

41.13.8 TCompressor.ZipBitFlag
Synopsis: Current bit
Declaration: function ZipBitFlag : Word; Virtual; Abstract
Visibility: public
Description: Abstract virtual function. Must be implemented in a descendent class.

41.13.9 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
### 41.13.10 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 (915), TShrinker.Compress (924), TDeflater.Compress (919)

### 41.13.11 TCompressor.OnPercent

**Synopsis:** Threshold percentage which triggers an OnProgress update

**Declaration:**
```
Property OnPercent : Integer
```

- **Visibility:** public
- **Access:** Read, Write

### 41.13.12 TCompressor.OnProgress

**Synopsis:** Event handler signalled to indicate the completion percentage for the compressor

**Declaration:**
```
Property OnProgress : TProgressEvent
```

- **Visibility:** public
- **Access:** Read, Write

### 41.13.13 TCompressor.Crc32Val

**Synopsis:** Running CRC32 value

**Declaration:**
```
Property Crc32Val : LongWord
```

- **Visibility:** public
- **Access:** Read, Write

**Description:** Running CRC32 value used when writing zip header.

### 41.13.14 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
41.14 **TDeCompressor**

41.14.1 **Description**

This object decompresses a compressed zip stream.

41.14.2 **Method overview**

<table>
<thead>
<tr>
<th>Page</th>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>917</td>
<td>Create</td>
<td>Creates decompressor object</td>
</tr>
<tr>
<td>917</td>
<td>DeCompress</td>
<td>Decompress zip stream</td>
</tr>
<tr>
<td>917</td>
<td>Terminate</td>
<td>Halts decompression and sets Terminated to True</td>
</tr>
<tr>
<td>918</td>
<td>ZipID</td>
<td>Identifier for type of compression</td>
</tr>
</tbody>
</table>

41.14.3 **Property overview**

<table>
<thead>
<tr>
<th>Page</th>
<th>Properties</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>918</td>
<td>BufferSize</td>
<td>r</td>
<td>Size of buffer used in decompression</td>
</tr>
<tr>
<td>918</td>
<td>Crc32Val</td>
<td>rw</td>
<td>Running CRC32 value used for verifying zip file integrity</td>
</tr>
<tr>
<td>918</td>
<td>OnPercent</td>
<td>rw</td>
<td>Percentage of decompression completion</td>
</tr>
<tr>
<td>918</td>
<td>OnProgress</td>
<td>rw</td>
<td>Event handler for OnProgress procedure</td>
</tr>
<tr>
<td>918</td>
<td>OnProgressEx</td>
<td>rw</td>
<td>Event handler signalled to indicate progress using processed and total byte counts</td>
</tr>
<tr>
<td>919</td>
<td>Terminated</td>
<td>r</td>
<td>Set to True when the Terminate method is called</td>
</tr>
</tbody>
</table>

41.14.4 **TDeCompressor.Create**

**Synopsis:** Creates decompressor object

**Declaration:**

```delphi
constructor Create(AInFile: TStream; AOutFile: TStream;
                   ABufSize: LongWord); Virtual
```

**Visibility:** public

41.14.5 **TDeCompressor.DeCompress**

**Synopsis:** Decompress zip stream

**Declaration:**

```delphi
procedure DeCompress; Virtual; Abstract
```

**Visibility:** public

41.14.6 **TDeCompressor.Terminate**

**Synopsis:** Halts decompression and sets Terminated to True

**Declaration:**

```delphi
procedure Terminate
```

**Visibility:** public

**Description:** Halts decompression and sets Terminated to True
41.14.7  TDeCompressor.ZipID
Synopsis: Identifier for type of compression
Declaration: class function ZipID : Word; Virtual; Abstract
Visibility: public

41.14.8  TDeCompressor.BufferSize
Synopsis: Size of buffer used in decompression
Declaration: Property BufferSize : LongWord
Visibility: public
Access: Read

41.14.9  TDeCompressor.OnPercent
Synopsis: Percentage of decompression completion
Declaration: Property OnPercent : Integer
Visibility: public
Access: Read,Write

41.14.10  TDeCompressor.OnProgress
Synopsis: Event handler for OnProgress procedure
Declaration: Property OnProgress : TProgressEvent
Visibility: public
Access: Read,Write

41.14.11  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

41.14.12  TDeCompressor.Crc32Val
Synopsis: Running CRC32 value used for verifying zip file integrity
Declaration: Property Crc32Val : LongWord
Visibility: public
Access: Read,Write
41.14.13  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

41.15  TDeflater

41.15.1  Description

Child of TCompressor (914) that implements the Deflate compression method

41.15.2  Method overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>919</td>
<td>Compress</td>
<td>Performs compression using the Deflate algorithm</td>
</tr>
<tr>
<td>919</td>
<td>Create</td>
<td>Constructor for the class instance</td>
</tr>
<tr>
<td>920</td>
<td>ZipBitFlag</td>
<td>Bitness flag</td>
</tr>
<tr>
<td>920</td>
<td>ZipID</td>
<td>Zip algorithm ID</td>
</tr>
<tr>
<td>920</td>
<td>ZipVersionReqd</td>
<td>Required version</td>
</tr>
</tbody>
</table>

41.15.3  Property overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Properties</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>920</td>
<td>CompressionLevel</td>
<td>rw</td>
<td>Indicates the compression level applied in the Compress method</td>
</tr>
</tbody>
</table>

41.15.4  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 (920), TCompressor.Create (915)

41.15.5  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.

### 41.15.6 TDeflater.ZipID

**Synopsis:** Zip algorithm ID

**Declaration:**
```pascal
class function ZipID : Word; Override
```

**Visibility:** public

**Description:** Zip algorithm ID

### 41.15.7 TDeflater.ZipVersionReqd

**Synopsis:** Required version

**Declaration:**
```pascal
class function ZipVersionReqd : Word; Override
```

**Visibility:** public

**Description:** Required version

### 41.15.8 TDeflater.ZipBitFlag

**Synopsis:** Bitness flag

**Declaration:**
```pascal
function ZipBitFlag : Word; Override
```

**Visibility:** public

**Description:** Bitness flag

### 41.15.9 TDeflater.CompressionLevel

**Synopsis:** Indicates the compression level applied in the Compress method

**Declaration:**
```pascal
Property CompressionLevel : Tcompressionlevel
```

**Visibility:** public

**Access:** Read, Write

**Description:** CompressionLevel is a TCompressionLevel property which Indicates the compression level applied in the Compress method. Values include:

- **clNone:** Do not use compression, just copy data.
- **clFastest:** Use the fast (but less) compression.
- **clDefault:** Use the default compression.
- **clMax:** Use the maximum compression.

**See also:** TDeflater.Compress (919), TCompressionLevel (944)
41.16  TFullZipFileEntries

41.16.1  Description
Collection of TFullZipFileEntry items

See also: TFullZipFileEntry (921)

41.16.2  Property overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Properties</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>921</td>
<td>FullEntries</td>
<td>rw</td>
<td>Array access to all entries</td>
</tr>
</tbody>
</table>

41.16.3  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 (921)

41.17  TFullZipFileEntry

41.17.1  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.

41.17.2  Property overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Properties</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>922</td>
<td>BitFlags</td>
<td>r</td>
<td>General purpose bit flag from the Local Header in the .ZIP archive file</td>
</tr>
<tr>
<td>922</td>
<td>CompressedSize</td>
<td>r</td>
<td>Size after applying the compression method and level</td>
</tr>
<tr>
<td>922</td>
<td>CompressMethod</td>
<td>r</td>
<td>Compression method for the file</td>
</tr>
<tr>
<td>922</td>
<td>CRC32</td>
<td>rw</td>
<td>32-bit CRC value for the file</td>
</tr>
</tbody>
</table>
41.17.3 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

41.17.4 TFullZipFileEntry.CompressMethod
Synopsis: Compression method for the file
Declaration: Property CompressMethod : Word
Visibility: public
Access: Read
Description: Compression method for the file

41.17.5 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

41.17.6 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

41.18 TInflater
41.18.1 Description
Child of TDeCompressor (917) that implements the Inflate decompression method
### 41.18.2 Method overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>923</td>
<td>Create</td>
<td>Constructor for the class instance</td>
</tr>
<tr>
<td>923</td>
<td>DeCompress</td>
<td>Removes compression applied using the deflate algorithm</td>
</tr>
<tr>
<td>923</td>
<td>ZipID</td>
<td>Zip algorithm ID</td>
</tr>
</tbody>
</table>

### 41.18.3 TInflater.Create

**Synopsis:** Constructor for the class instance

**Declaration:**

```pascal
constructor Create(AInFile: TStream; AOutFile: TStream; ABufSize: LongWord); Override
```

**Visibility:** public

**Description:** Constructor for the class instance

### 41.18.4 TInflater.DeCompress

**Synopsis:** Removes compression applied using the deflate algorithm

**Declaration:**

```pascal
procedure DeCompress; Override
```

**Visibility:** public

**Description:** Removes compression applied using the deflate algorithm

### 41.18.5 TInflater.ZipID

**Synopsis:** Zip algorithm ID

**Declaration:**

```pascal
class function ZipID : Word; Override
```

**Visibility:** public

**Description:** Zip algorithm ID

### 41.19 TShrinker

#### 41.19.1 Description

TShrinker implements the LZW lossless data compression algorithm created by Abraham Lempel, Jacob Ziv, and Terry Welch also known as "shrink" compression.

#### 41.19.2 Method overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>924</td>
<td>Compress</td>
<td>Compresses input values using LZW (shrink) compression</td>
</tr>
<tr>
<td>924</td>
<td>Create</td>
<td>Constructor for the class instance</td>
</tr>
<tr>
<td>924</td>
<td>Destroy</td>
<td>Destructor for the class instance</td>
</tr>
<tr>
<td>924</td>
<td>ZipBitFlag</td>
<td>Zip bitness flag</td>
</tr>
<tr>
<td>924</td>
<td>ZipID</td>
<td>Return Zip algorithm ID</td>
</tr>
<tr>
<td>924</td>
<td>ZipVersionReqd</td>
<td>Minimum zip algorithm required</td>
</tr>
</tbody>
</table>
41.19.3 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

41.19.4 TShrinker.Destroy
Synopsis: Destructor for the class instance

Declaration: destructor Destroy; Override

Visibility: public

Description: Destructor for the class instance

41.19.5 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.

41.19.6 TShrinker.ZipID
Synopsis: Return Zip algorithm ID

Declaration: class function ZipID : Word; Override

Visibility: public

Description: Return Zip algorithm ID

41.19.7 TShrinker.ZipVersionReqd
Synopsis: Minimum zip algorithm required

Declaration: class function ZipVersionReqd : Word; Override

Visibility: public

Description: Minimum zip algorithm required

41.19.8 TShrinker.ZipBitFlag
Synopsis: Zip bitness flag

Declaration: function ZipBitFlag : Word; Override

Visibility: public

Description: Zip bitness flag
41.20  TUnZipper

41.20.1  Description

Extracts and decompresses files and directories in a .ZIP archive file

41.20.2  Method overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>927</td>
<td>Clear</td>
<td>Removes all entries and files from object</td>
</tr>
<tr>
<td>926</td>
<td>Create</td>
<td>Constructor for the class instance</td>
</tr>
<tr>
<td>926</td>
<td>Destroy</td>
<td>Destructor for the class instance</td>
</tr>
<tr>
<td>927</td>
<td>Examine</td>
<td>Opens zip file and reads the directory entries (list of zipped files)</td>
</tr>
<tr>
<td>928</td>
<td>Terminate</td>
<td>Sets the value in Terminated to True</td>
</tr>
<tr>
<td>927</td>
<td>Unzip</td>
<td>Unzips the specified .ZIP archive file</td>
</tr>
<tr>
<td>926</td>
<td>UnZipAllFiles</td>
<td>Unzips all files in a zip file, writing them to disk</td>
</tr>
<tr>
<td>926</td>
<td>UnZipFile</td>
<td>Unzips a single file found in the specified .ZIP archive</td>
</tr>
<tr>
<td>927</td>
<td>UnZipFiles</td>
<td>Unzips the specified files in a .ZIP archive file</td>
</tr>
</tbody>
</table>

41.20.3  Property overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Properties</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>928</td>
<td>BufferSize</td>
<td>rw</td>
<td>Size of the buffer used to read and decompress entries in the .ZIP file</td>
</tr>
<tr>
<td>931</td>
<td>Entries</td>
<td>r</td>
<td>Collection with TFullZipFileEntry instances for files and directories stored in the .ZIP archive</td>
</tr>
<tr>
<td>930</td>
<td>FileComment</td>
<td>r</td>
<td>Comment stored in the .ZIP archive file</td>
</tr>
<tr>
<td>930</td>
<td>FileName</td>
<td>rw</td>
<td>Path and file name for the .zip file to be unzipped / processed</td>
</tr>
<tr>
<td>930</td>
<td>Files</td>
<td>r</td>
<td>Files in the zip file (deprecated)</td>
</tr>
<tr>
<td>931</td>
<td>Flat</td>
<td>rw</td>
<td>Extracts files to a single directory</td>
</tr>
<tr>
<td>928</td>
<td>OnCloseInputStream</td>
<td>rw</td>
<td>Event handler signalled when the input stream for the .ZIP file is closed</td>
</tr>
<tr>
<td>928</td>
<td>OnCreateStream</td>
<td>rw</td>
<td>Event handler signalled when an output stream is created</td>
</tr>
<tr>
<td>929</td>
<td>OnDoneStream</td>
<td>rw</td>
<td>Event handler signalled when an output stream is closed</td>
</tr>
<tr>
<td>930</td>
<td>OnEndFile</td>
<td>rw</td>
<td>Callback procedure that will be called after unzipping a file</td>
</tr>
<tr>
<td>928</td>
<td>OnOpenInputStream</td>
<td>rw</td>
<td>Event handler signalled when the input stream for the .ZIP file is opened</td>
</tr>
<tr>
<td>929</td>
<td>OnPercent</td>
<td>rw</td>
<td>Threshold percentage which triggers a progress notification</td>
</tr>
<tr>
<td>929</td>
<td>OnProgress</td>
<td>rw</td>
<td>Progress event handler used when decompressing files</td>
</tr>
<tr>
<td>929</td>
<td>OnProgressEx</td>
<td>rw</td>
<td>Extended progress event handler used when decompressing files</td>
</tr>
<tr>
<td>929</td>
<td>OnStartFile</td>
<td>rw</td>
<td>Callback procedure that will be called before unzipping a file</td>
</tr>
<tr>
<td>930</td>
<td>OutputPath</td>
<td>rw</td>
<td>Path where archive files will be unzipped</td>
</tr>
<tr>
<td>932</td>
<td>Terminated</td>
<td>r</td>
<td>True if the Terminate method has been called</td>
</tr>
<tr>
<td>931</td>
<td>UseUTF8</td>
<td>rw</td>
<td>Indicates that the UTF-8-encoded names are used when locating and unzipping entries in the archive</td>
</tr>
</tbody>
</table>

925
CHAPTER 41. REFERENCE FOR UNIT 'ZIPPER'

41.20.4 TUnZipper.Create
Synopsis: Constructor for the class instance
Declaration: constructor Create
Visibility: public
Description: Constructor for the class instance

41.20.5 TUnZipper.Destroy
Synopsis: Destructor for the class instance
Declaration: destructor Destroy; Override
Visibility: public
Description: Destructor for the class instance

41.20.6 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 (937) 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.

41.20.7 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
41.20.8 TUnZipper.UnZipFiles

Synopsis: Unzips the specified files in a .ZIP archive file

Declaration:

```delphi
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

41.20.9 TUnZipper.Unzip

Synopsis: Unzips the specified .ZIP archive file

Declaration:

```delphi
class procedure Unzip(const AZipFileName: RawByteString)
class procedure Unzip(const AZipFileName: RawByteString;
  aExtractFileName: RawByteString)
class procedure Unzip(const AZipFileName: RawByteString;
  aFileList: Array of RawByteString)
class procedure Unzip(const AZipFileName: RawByteString;
  aFileList: TStrings)
```

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.

41.20.10 TUnZipper.Clear

Synopsis: Removes all entries and files from object

Declaration: procedure Clear

Visibility: public

Description: Removes all entries and files from object

41.20.11 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)
41.20.12 TUnZipper.Terminate

Synopsis: Sets the value in Terminated to True

Declaration: procedure Terminate

Visibility: public

Description: Sets the value in Terminated to True

41.20.13 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

41.20.14 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

41.20.15 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

41.20.16 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
41.20.17  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

41.20.18  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

41.20.19  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

41.20.20  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

41.20.21  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
41.20.22  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

41.20.23  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

41.20.24  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

41.20.25  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

41.20.26  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.
41.20.27  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.


41.20.28  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 (931), TUnZipper.UnZipAllFiles (926), TUnZipper.UnZipFiles (927), TUnZipper.UnZipFile (926), TZipFileEntries (932), TZipFileEntry (933)

41.20.29  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.
41.20.30  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

41.21  TZipFileEntries

41.21.1  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 (933), TZipFileEntry (933), TZipper.Entries (943), TUnZipper.Entries (931), TFullZipFileEntries (921)

41.21.2  Method overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>933</td>
<td>AddFileEntries</td>
<td>Adds TZipFileEntry instances in the collection for the file names in List</td>
</tr>
<tr>
<td>932</td>
<td>AddFileEntry</td>
<td>Adds file to zip directory</td>
</tr>
</tbody>
</table>

41.21.3  Property overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Properties</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>933</td>
<td>Entries</td>
<td>rw</td>
<td>Entries (files) in the zip archive</td>
</tr>
</tbody>
</table>

41.21.4  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.

932
41.21.5  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 (932)

41.21.6  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 (933), TCollection.Items (??)

41.22  TZipFileEntry

41.22.1  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.

### 41.22.2 Method overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>935</td>
<td>Assign</td>
<td>Copies property values from the specified persistent object</td>
</tr>
<tr>
<td>934</td>
<td>Create</td>
<td>Constructor for the class instance</td>
</tr>
<tr>
<td>934</td>
<td>IsDirectory</td>
<td>True if the entry is a directory on the local file system</td>
</tr>
<tr>
<td>935</td>
<td>IsLink</td>
<td>True if the directory is a symbolic link on the local file system</td>
</tr>
</tbody>
</table>

### 41.22.3 Property overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Properties</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>935</td>
<td>ArchiveFileName</td>
<td>rw</td>
<td>Name of the file or directory in the .ZIP archive</td>
</tr>
<tr>
<td>937</td>
<td>Attributes</td>
<td>rw</td>
<td>File attributes for the file or directory</td>
</tr>
<tr>
<td>937</td>
<td>CompressionLevel</td>
<td>rw</td>
<td>Compression level applied to the content stored in the .ZIP archive</td>
</tr>
<tr>
<td>936</td>
<td>DateTime</td>
<td>rw</td>
<td>Timestamp for the file or directory in the .ZIP archive</td>
</tr>
<tr>
<td>936</td>
<td>DiskFileName</td>
<td>rw</td>
<td>Name of the file or directory on the local file system</td>
</tr>
<tr>
<td>936</td>
<td>OS</td>
<td>rw</td>
<td>Indication of operating system/file system</td>
</tr>
<tr>
<td>936</td>
<td>Size</td>
<td>rw</td>
<td>Size of the compressed content for the file or directory</td>
</tr>
<tr>
<td>935</td>
<td>Stream</td>
<td>rw</td>
<td>Stream with the content for the entry</td>
</tr>
<tr>
<td>935</td>
<td>UTF8ArchiveFileName</td>
<td>rw</td>
<td>Name of the file or directory in the .ZIP archive using UTF-8 encoding</td>
</tr>
<tr>
<td>936</td>
<td>UTF8DiskFileName</td>
<td>rw</td>
<td>Name of the file or directory on the local file system using UTF-8 encoding</td>
</tr>
</tbody>
</table>

### 41.22.4 TZipFileEntry.Create

**Synopsis:** Constructor for the class instance

**Declaration:** `constructor Create(ACollection: TCollection); Override`

**Visibility:** public

**Description:** `Create` is the overridden constructor for the class instance. `Create` sets the default values for properties, including:

- **DateTime** Sets to the current date and time for the local computer.
- **OS** Set to OS_UNIX for UNIX-like environments, or OS_VFAT for all others.
- **Attributes** Set to 0 (no attributes).
- **CompressionLevel** Set to clDefault.

Create calls the inherited constructor prior to exiting from the method.

### 41.22.5 TZipFileEntry.IsDirectory

**Synopsis:** True if the entry is a directory on the local file system
41.22.6  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

41.22.7  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

41.22.8  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

41.22.9  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

41.22.10  TZipFileEntry.UTF8ArchiveFileName

Declaration: Property UTF8ArchiveFileName : UTF8String

Visibility: published

Access: Read, Write
41.22.11  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

41.22.12  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

41.22.13  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

41.22.14  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

41.22.15  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.
41.22.16 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

41.22.17 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

41.23 TZipper

41.23.1 Description

Creates a .ZIP archive file

41.23.2 Method overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>940</td>
<td>Clear</td>
<td>Removes all values in the Entries and Files properties</td>
</tr>
<tr>
<td>938</td>
<td>Create</td>
<td>Constructor for the class instance</td>
</tr>
<tr>
<td>938</td>
<td>Destroy</td>
<td>Destructor for the class instance</td>
</tr>
<tr>
<td>939</td>
<td>SaveToFile</td>
<td>Saves the archive to a file with a new name</td>
</tr>
<tr>
<td>939</td>
<td>SaveToStream</td>
<td>Save the archive to a stream</td>
</tr>
<tr>
<td>941</td>
<td>Terminate</td>
<td>Halts an assigned compressor in the class instance, and sets Terminated to True</td>
</tr>
<tr>
<td>940</td>
<td>Zip</td>
<td>Convenience method used to create a .zip file with the given name containing the specified file(s)</td>
</tr>
<tr>
<td>938</td>
<td>ZipAllFiles</td>
<td>Zips all files in object and writes zip to disk</td>
</tr>
<tr>
<td>939</td>
<td>ZipFile</td>
<td>Zip one file to a zip file</td>
</tr>
<tr>
<td>939</td>
<td>ZipFiles</td>
<td>Zip multiple files into an archive</td>
</tr>
</tbody>
</table>
### 41.23.3 Property overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Properties</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>941</td>
<td>BufferSize</td>
<td>rw</td>
<td>Buffer size used when reading and processing files</td>
</tr>
<tr>
<td>943</td>
<td>Entries</td>
<td>rw</td>
<td>Collection with the TZipFileEntry instances in the .ZIP archive</td>
</tr>
<tr>
<td>942</td>
<td>FileComment</td>
<td>rw</td>
<td>Comment stored in the .ZIP archive file</td>
</tr>
<tr>
<td>942</td>
<td>FileName</td>
<td>rw</td>
<td>Name of the .ZIP archive file where the compressed files and directories are stored</td>
</tr>
<tr>
<td>942</td>
<td>Files</td>
<td>r</td>
<td>Provides access to the list of files and directories in the archive</td>
</tr>
<tr>
<td>942</td>
<td>InMemSize</td>
<td>rw</td>
<td>Total memory used for the compressed content in the .ZIP file</td>
</tr>
<tr>
<td>942</td>
<td>OnEndFile</td>
<td>rw</td>
<td>Event handler signalled when compression for a file has been completed</td>
</tr>
<tr>
<td>944</td>
<td>OnPercent</td>
<td>rw</td>
<td>Threshold percentage which triggers progress notifications when processing files</td>
</tr>
<tr>
<td>941</td>
<td>OnProgress</td>
<td>rw</td>
<td>Event handler signalled to show a percent complete progress notifications</td>
</tr>
<tr>
<td>941</td>
<td>OnStartFile</td>
<td>rw</td>
<td>Event handler signalled when compression for a file is started</td>
</tr>
<tr>
<td>943</td>
<td>Terminated</td>
<td>r</td>
<td>True if the Terminate method has been called</td>
</tr>
<tr>
<td>943</td>
<td>UseLanguageEncoding</td>
<td>rw</td>
<td>Use language encoding</td>
</tr>
</tbody>
</table>

### 41.23.4 TZipper.Create

**Synopsis:** Constructor for the class instance

**Declaration:** constructor Create

**Visibility:** public

**Description:** Constructor for the class instance

### 41.23.5 TZipper.Destroy

**Synopsis:** Destructor for the class instance

**Declaration:** destructor Destroy; Override

**Visibility:** public

**Description:** Destructor for the class instance

### 41.23.6 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 (937) 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 zip file name
    for I := 2 to ParamCount do //Use the other arguments on the command line as files to be zipped
      Zipper.Entries.AddFileEntry(ParamStr(I), ParamStr(I));
    Zipper.ZipAllFiles;
  finally
    Zipper.Free;
  end;
end.

41.23.7 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.

41.23.8 TZipper.SaveToStream
Synopsis: Save the archive to a stream

Declaration: procedure SaveToStream(AStream: TStream)

Visibility: public

Description: Save the archive to a stream

41.23.9 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 (939)

41.23.10 TZipper.ZipFiles
Synopsis: Zip multiple files into an archive
CHAPTER 41. REFERENCE FOR UNIT 'ZIPPER'

Declaration:

```delphi
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)
class procedure Zip(const AZipFileName: RawByteString;
    Entries: TZipFileEntries)
class procedure Zip(const aFileList: Array of RawbyteString)
class procedure Zip(Entries: TZipFileEntries)
```

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:

```delphi
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 (927)

41.23.12 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
41.23.13  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

41.23.14  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

41.23.15  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

41.23.16  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

41.23.17  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
41.23.18 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

41.23.19 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

41.23.20 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

41.23.21 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 (943), TZipperFileEntries.AddFileEntry (906), TZipperFileEntries.AddFileEntries (906)

41.23.22 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
41.23.23  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

41.23.24  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

41.23.25  TZipper.UseLanguageEncoding
Synopsis: Use language encoding

Declaration: Property UseLanguageEncoding : Boolean
  Visibility: public
  Access: Read, Write

Description: EFS/language encoding using UTF-8.
Chapter 42

Reference for unit ’ZStream’

42.1 Used units

Table 42.1: Used units by unit ’ZStream’

<table>
<thead>
<tr>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Classes</td>
<td>??</td>
</tr>
<tr>
<td>gzip</td>
<td>??</td>
</tr>
<tr>
<td>System</td>
<td>??</td>
</tr>
<tr>
<td>zbase</td>
<td>??</td>
</tr>
</tbody>
</table>

42.2 Overview

The ZStream unit implements a TStream (??) descendent (TCompressionStream (945)) 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 (948)) and decompresses it with the inflate algorithm.

42.3 Constants, types and variables

42.3.1 Types

Tcompressionlevel = (cldone,clfastecldefault,clmax)

Table 42.2: Enumeration values for type Tcompressionlevel

<table>
<thead>
<tr>
<th>Value</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>cldone</td>
<td>Use default compression</td>
</tr>
<tr>
<td>clfastecldefault</td>
<td>Use fast (but less) compression.</td>
</tr>
<tr>
<td>clmax</td>
<td>Use maximum compression</td>
</tr>
<tr>
<td>cldone</td>
<td>Do not use compression, just copy data.</td>
</tr>
</tbody>
</table>
Compression level for the deflate algorithm

Tgzopenmode = (gzopenread, gzopenwrite)

<table>
<thead>
<tr>
<th>Value</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>gzopenread</td>
<td>Open file for reading</td>
</tr>
<tr>
<td>gzopenwrite</td>
<td>Open file for writing</td>
</tr>
</tbody>
</table>

Open mode for gzip file.

42.4 Ecompressionerror

42.4.1 Description

ECompressionError is the exception class used by the TCompressionStream (945) class.

42.5 Edecompressionerror

42.5.1 Description

EDecompressionError is the exception class used by the TDeCompressionStream (948) class.

42.6 Egzfileerror

42.6.1 Description

Egzfileerror is the exception class used to report errors by the Tgzfilestream (951) class.

See also: Tgzfilestream (951)

42.7 Ezliberror

42.7.1 Description

Errors which occur in the zstream unit are signaled by raising an EZLibError exception descendant.

42.8 Tcompressionstream

42.8.1 Description

TCompressionStream
CHAPTER 42. REFERENCE FOR UNIT 'ZSTREAM'

42.8.2 Method overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>946</td>
<td>create</td>
<td>Create a new instance of the compression stream.</td>
</tr>
<tr>
<td>946</td>
<td>destroy</td>
<td>Flushes data to the output stream and destroys the compression stream.</td>
</tr>
<tr>
<td>947</td>
<td>flush</td>
<td>Flush remaining data to the target stream</td>
</tr>
<tr>
<td>947</td>
<td>_compressionrate</td>
<td>Get the current compression rate</td>
</tr>
<tr>
<td>946</td>
<td>write</td>
<td>Write data to the stream</td>
</tr>
</tbody>
</table>

42.8.3 Property overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Properties</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>947</td>
<td>OnProgress</td>
<td></td>
<td>Progress handler</td>
</tr>
</tbody>
</table>

42.8.4 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 (946)

42.8.5 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 (946)

42.8.6 Tcompressionstream.write

Synopsis: Write data to the stream

Declaration: function write(const buffer; count: LongInt) : LongInt; Override

Visibility: public
CHAPTER 42. REFERENCE FOR UNIT ‘ZSTREAM’

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 (945) exception is raised.

See also: Write (946), ECompressionError (945)

42.8.7 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.

42.8.8 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.

42.8.9 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.

42.9 Tcustomzlibstream

42.9.1 Description
TCustomZlibStream serves as the ancestor class for the TCompressionStream (945) and TDecompressionStream (948) classes.

It introduces support for a progress handler, and stores the input or output stream.
42.9.2  Method overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>948</td>
<td>create</td>
<td>Create a new instance of TCustomZlibStream</td>
</tr>
<tr>
<td>948</td>
<td>destroy</td>
<td>Clear up instance</td>
</tr>
</tbody>
</table>

42.9.3  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 (945), TDecompressionStream (948)

42.9.4  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 (948)

42.10  Tdecompressionstream

42.10.1  Description

TDecompressionStream performs the inverse operation of TCompressionStream (945). 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 (945) compression stream.

See also: TCompressionStream (945)

42.10.2  Method overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>949</td>
<td>create</td>
<td>Creates a new instance of the TDecompressionStream stream</td>
</tr>
<tr>
<td>949</td>
<td>destroy</td>
<td>Destroys the TDecompressionStream instance</td>
</tr>
<tr>
<td>950</td>
<td>get_compressionrate</td>
<td>Get the current compression rate</td>
</tr>
<tr>
<td>949</td>
<td>read</td>
<td>Read data from the compressed stream</td>
</tr>
<tr>
<td>950</td>
<td>Seek</td>
<td>Move stream position to a certain location in the stream.</td>
</tr>
</tbody>
</table>
CHAPTER 42. REFERENCE FOR UNIT ‘ZSTREAM’

42.10.3 Property overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Properties</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>950</td>
<td>OnProgress</td>
<td></td>
<td>Progress handler</td>
</tr>
</tbody>
</table>

42.10.4 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 (949)

42.10.5 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 (949)

42.10.6 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 (945) exception is raised.

See also: Write (946)

949
42.10.7 Tdecompressionstream.Seek

Synopsis: Move stream position to a certain location in the stream.

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

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 (945) exception is raised if the stream does not allow the requested seek operation.

See also: Read (949)

42.10.8 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.

42.10.9 Tdecompressionstream.OnProgress

Synopsis: Progress handler

Declaration: Property OnProgress :

Visibility: public

Access: public

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.
42.11 TGZFileStream

42.11.1 Description

TGZFileStream can be used to read data from a gzip file, or to write data to a gzip file.

See also: TCompressionStream (945), TDeCompressionStream (948)

42.11.2 Method overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>951</td>
<td>create</td>
<td>Create a new instance of TGZFileStream</td>
</tr>
<tr>
<td>952</td>
<td>destroy</td>
<td>Removes TGZFileStream instance</td>
</tr>
<tr>
<td>951</td>
<td>read</td>
<td>Read data from the compressed file</td>
</tr>
<tr>
<td>952</td>
<td>seek</td>
<td>Set the position in the compressed stream.</td>
</tr>
<tr>
<td>952</td>
<td>write</td>
<td>Write data to be compressed</td>
</tr>
</tbody>
</table>

42.11.3 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 (951) 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 (952) method will be stored in the file in compressed (deflated) form.

Errors: If the file is not found, an EZlibError (945) exception is raised.

See also: Destroy (952), TGZOpenMode (945)

42.11.4 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 (951), Write (952), Seek (952)
42.11.5 TGZFileStream.write

**Synopsis:** Write data to be compressed

**Declaration:**

```pascal
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 (945) exception is raised.

See also: Create (951), Read (951), Seek (952)

42.11.6 TGZFileStream.seek

**Synopsis:** Set the position in the compressed stream.

**Declaration:**

```pascal
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 (950) for a list of possibilities.

**Errors:** In case an impossible combination is asked, an EZlibError (945) exception is raised.

See also: TDecompressionStream.Seek (950)

42.11.7 TGZFileStream.destroy

**Synopsis:** Removes TGZFileStream instance

**Declaration:**

```pascal
destructor destroy; Override
```

**Visibility:** public

**Description:**

Destroy closes the file and releases the TGZFileStream instance from memory.

See also: Create (951)