

Free Component Library (FCL):  
Reference guide.

---

Reference guide for FCL units.  
Document version 2.6  
March 2014

---

Michaël Van Canneyt

---

# Contents

0.1	Overview	70
<b>1</b>	<b>Reference for unit 'ascii85'</b>	<b>71</b>
1.1	Used units	71
1.2	Overview	71
1.3	Constants, types and variables	71
1.3.1	Types	71
1.4	TASCII85DecoderStream	72
1.4.1	Description	72
1.4.2	Method overview	72
1.4.3	Property overview	72
1.4.4	TASCII85DecoderStream.Create	72
1.4.5	TASCII85DecoderStream.Decode	73
1.4.6	TASCII85DecoderStream.Close	73
1.4.7	TASCII85DecoderStream.ClosedP	73
1.4.8	TASCII85DecoderStream.Destroy	73
1.4.9	TASCII85DecoderStream.Read	74
1.4.10	TASCII85DecoderStream.Seek	74
1.4.11	TASCII85DecoderStream.BExpectBoundary	74
1.5	TASCII85EncoderStream	74
1.5.1	Description	74
1.5.2	Method overview	75
1.5.3	Property overview	75
1.5.4	TASCII85EncoderStream.Create	75
1.5.5	TASCII85EncoderStream.Destroy	75
1.5.6	TASCII85EncoderStream.Write	75
1.5.7	TASCII85EncoderStream.Width	76
1.5.8	TASCII85EncoderStream.Boundary	76
1.6	TASCII85RingBuffer	76
1.6.1	Description	76
1.6.2	Method overview	76

1.6.3	Property overview . . . . .	76
1.6.4	TASCII85RingBuffer.Write . . . . .	77
1.6.5	TASCII85RingBuffer.Read . . . . .	77
1.6.6	TASCII85RingBuffer.FillCount . . . . .	77
1.6.7	TASCII85RingBuffer.Size . . . . .	77
<b>2</b>	<b>Reference for unit 'AVL_Tree'</b>	<b>78</b>
2.1	Used units . . . . .	78
2.2	Overview . . . . .	78
2.3	TAVLTree . . . . .	78
2.3.1	Description . . . . .	78
2.3.2	Method overview . . . . .	79
2.3.3	Property overview . . . . .	79
2.3.4	TAVLTree.Find . . . . .	79
2.3.5	TAVLTree.FindKey . . . . .	80
2.3.6	TAVLTree.FindSuccessor . . . . .	80
2.3.7	TAVLTree.FindPrecessor . . . . .	80
2.3.8	TAVLTree.FindLowest . . . . .	80
2.3.9	TAVLTree.FindHighest . . . . .	81
2.3.10	TAVLTree.FindNearest . . . . .	81
2.3.11	TAVLTree.FindPointer . . . . .	81
2.3.12	TAVLTree.FindLeftMost . . . . .	81
2.3.13	TAVLTree.FindRightMost . . . . .	82
2.3.14	TAVLTree.FindLeftMostKey . . . . .	82
2.3.15	TAVLTree.FindRightMostKey . . . . .	82
2.3.16	TAVLTree.FindLeftMostSameKey . . . . .	82
2.3.17	TAVLTree.FindRightMostSameKey . . . . .	83
2.3.18	TAVLTree.Add . . . . .	83
2.3.19	TAVLTree.Delete . . . . .	83
2.3.20	TAVLTree.Remove . . . . .	83
2.3.21	TAVLTree.RemovePointer . . . . .	84
2.3.22	TAVLTree.MoveDataLeftMost . . . . .	84
2.3.23	TAVLTree.MoveDataRightMost . . . . .	84
2.3.24	TAVLTree.Clear . . . . .	84
2.3.25	TAVLTree.FreeAndClear . . . . .	85
2.3.26	TAVLTree.FreeAndDelete . . . . .	85
2.3.27	TAVLTree.ConsistencyCheck . . . . .	85
2.3.28	TAVLTree.WriteReportToStream . . . . .	85
2.3.29	TAVLTree.ReportAsString . . . . .	86
2.3.30	TAVLTree.SetNodeManager . . . . .	86

2.3.31	TAVLTree.Create . . . . .	86
2.3.32	TAVLTree.Destroy . . . . .	86
2.3.33	TAVLTree.GetEnumerator . . . . .	87
2.3.34	TAVLTree.OnCompare . . . . .	87
2.3.35	TAVLTree.Count . . . . .	87
2.4	TAVLTreeNode . . . . .	87
2.4.1	Description . . . . .	87
2.4.2	Method overview . . . . .	87
2.4.3	TAVLTreeNode.Clear . . . . .	88
2.4.4	TAVLTreeNode.TreeDepth . . . . .	88
2.5	TAVLTreeNodeEnumerator . . . . .	88
2.5.1	Description . . . . .	88
2.5.2	Method overview . . . . .	88
2.5.3	Property overview . . . . .	88
2.5.4	TAVLTreeNodeEnumerator.Create . . . . .	88
2.5.5	TAVLTreeNodeEnumerator.MoveNext . . . . .	89
2.5.6	TAVLTreeNodeEnumerator.Current . . . . .	89
2.6	TAVLTreeNodeMemManager . . . . .	89
2.6.1	Description . . . . .	89
2.6.2	Method overview . . . . .	89
2.6.3	Property overview . . . . .	89
2.6.4	TAVLTreeNodeMemManager.DisposeNode . . . . .	90
2.6.5	TAVLTreeNodeMemManager.NewNode . . . . .	90
2.6.6	TAVLTreeNodeMemManager.Clear . . . . .	90
2.6.7	TAVLTreeNodeMemManager.Create . . . . .	90
2.6.8	TAVLTreeNodeMemManager.Destroy . . . . .	90
2.6.9	TAVLTreeNodeMemManager.MinimumFreeNode . . . . .	91
2.6.10	TAVLTreeNodeMemManager.MaximumFreeNodeRatio . . . . .	91
2.6.11	TAVLTreeNodeMemManager.Count . . . . .	91
2.7	TBaseAVLTreeNodeManager . . . . .	91
2.7.1	Description . . . . .	91
2.7.2	Method overview . . . . .	92
2.7.3	TBaseAVLTreeNodeManager.DisposeNode . . . . .	92
2.7.4	TBaseAVLTreeNodeManager.NewNode . . . . .	92
3	Reference for unit 'base64'	93
3.1	Used units . . . . .	93
3.2	Overview . . . . .	93
3.3	Constants, types and variables . . . . .	93
3.3.1	Types . . . . .	93

3.4	Procedures and functions . . . . .	94
3.4.1	DecodeStringBase64 . . . . .	94
3.4.2	EncodeStringBase64 . . . . .	94
3.5	EBase64DecodingException . . . . .	94
3.5.1	Description . . . . .	94
3.6	TBase64DecodingStream . . . . .	94
3.6.1	Description . . . . .	94
3.6.2	Method overview . . . . .	95
3.6.3	Property overview . . . . .	95
3.6.4	TBase64DecodingStream.Create . . . . .	95
3.6.5	TBase64DecodingStream.Reset . . . . .	95
3.6.6	TBase64DecodingStream.Read . . . . .	95
3.6.7	TBase64DecodingStream.Seek . . . . .	96
3.6.8	TBase64DecodingStream.EOF . . . . .	96
3.6.9	TBase64DecodingStream.Mode . . . . .	96
3.7	TBase64EncodingStream . . . . .	97
3.7.1	Description . . . . .	97
3.7.2	Method overview . . . . .	97
3.7.3	TBase64EncodingStream.Destroy . . . . .	97
3.7.4	TBase64EncodingStream.Flush . . . . .	97
3.7.5	TBase64EncodingStream.Write . . . . .	98
3.7.6	TBase64EncodingStream.Seek . . . . .	98
<b>4</b>	<b>Reference for unit 'BlowFish'</b>	<b>99</b>
4.1	Used units . . . . .	99
4.2	Overview . . . . .	99
4.3	Constants, types and variables . . . . .	99
4.3.1	Constants . . . . .	99
4.3.2	Types . . . . .	99
4.4	EBlowFishError . . . . .	100
4.4.1	Description . . . . .	100
4.5	TBlowFish . . . . .	100
4.5.1	Description . . . . .	100
4.5.2	Method overview . . . . .	100
4.5.3	TBlowFish.Create . . . . .	100
4.5.4	TBlowFish.Encrypt . . . . .	101
4.5.5	TBlowFish.Decrypt . . . . .	101
4.6	TBlowFishDeCryptStream . . . . .	101
4.6.1	Description . . . . .	101
4.6.2	Method overview . . . . .	101

4.6.3	TBlowFishDeCryptStream.Read . . . . .	101
4.6.4	TBlowFishDeCryptStream.Seek . . . . .	102
4.7	TBlowFishEncryptStream . . . . .	102
4.7.1	Description . . . . .	102
4.7.2	Method overview . . . . .	102
4.7.3	TBlowFishEncryptStream.Destroy . . . . .	102
4.7.4	TBlowFishEncryptStream.Write . . . . .	103
4.7.5	TBlowFishEncryptStream.Seek . . . . .	103
4.7.6	TBlowFishEncryptStream.Flush . . . . .	103
4.8	TBlowFishStream . . . . .	104
4.8.1	Description . . . . .	104
4.8.2	Method overview . . . . .	104
4.8.3	Property overview . . . . .	104
4.8.4	TBlowFishStream.Create . . . . .	104
4.8.5	TBlowFishStream.Destroy . . . . .	104
4.8.6	TBlowFishStream.BlowFish . . . . .	105
<b>5</b>	<b>Reference for unit 'bufstream'</b>	<b>106</b>
5.1	Used units . . . . .	106
5.2	Overview . . . . .	106
5.3	Constants, types and variables . . . . .	106
5.3.1	Constants . . . . .	106
5.4	TBufStream . . . . .	106
5.4.1	Description . . . . .	106
5.4.2	Method overview . . . . .	107
5.4.3	Property overview . . . . .	107
5.4.4	TBufStream.Create . . . . .	107
5.4.5	TBufStream.Destroy . . . . .	107
5.4.6	TBufStream.Buffer . . . . .	108
5.4.7	TBufStream.Capacity . . . . .	108
5.4.8	TBufStream.BufferPos . . . . .	108
5.4.9	TBufStream.BufferSize . . . . .	108
5.5	TReadBufStream . . . . .	109
5.5.1	Description . . . . .	109
5.5.2	Method overview . . . . .	109
5.5.3	TReadBufStream.Seek . . . . .	109
5.5.4	TReadBufStream.Read . . . . .	109
5.6	TWriteBufStream . . . . .	110
5.6.1	Description . . . . .	110
5.6.2	Method overview . . . . .	110

5.6.3	TWriteBufStream.Destroy	110
5.6.4	TWriteBufStream.Seek	110
5.6.5	TWriteBufStream.Write	110
<b>6</b>	<b>Reference for unit 'CacheCls'</b>	<b>112</b>
6.1	Used units	112
6.2	Overview	112
6.3	Constants, types and variables	112
6.3.1	Resource strings	112
6.3.2	Types	112
6.4	ECacheError	113
6.4.1	Description	113
6.5	TCache	113
6.5.1	Description	113
6.5.2	Method overview	114
6.5.3	Property overview	114
6.5.4	TCache.Create	114
6.5.5	TCache.Destroy	114
6.5.6	TCache.Add	114
6.5.7	TCache.AddNew	115
6.5.8	TCache.FindSlot	115
6.5.9	TCache.IndexOf	115
6.5.10	TCache.Remove	116
6.5.11	TCache.Data	116
6.5.12	TCache.MRUSlot	116
6.5.13	TCache.LRUSlot	117
6.5.14	TCache.SlotCount	117
6.5.15	TCache.Slots	117
6.5.16	TCache.OnIsDataEqual	117
6.5.17	TCache.OnFreeSlot	118
<b>7</b>	<b>Reference for unit 'contnrs'</b>	<b>119</b>
7.1	Used units	119
7.2	Overview	119
7.3	Constants, types and variables	119
7.3.1	Constants	119
7.3.2	Types	120
7.4	Procedures and functions	123
7.4.1	RSHash	123
7.5	EDuplicate	123
7.5.1	Description	123

7.6	EKeyNotFound . . . . .	123
7.6.1	Description . . . . .	123
7.7	TBucketList . . . . .	123
7.7.1	Description . . . . .	123
7.7.2	Method overview . . . . .	124
7.7.3	TBucketList.Create . . . . .	124
7.8	TClassList . . . . .	124
7.8.1	Description . . . . .	124
7.8.2	Method overview . . . . .	124
7.8.3	Property overview . . . . .	124
7.8.4	TClassList.Add . . . . .	125
7.8.5	TClassList.Extract . . . . .	125
7.8.6	TClassList.Remove . . . . .	125
7.8.7	TClassList.IndexOf . . . . .	125
7.8.8	TClassList.First . . . . .	126
7.8.9	TClassList.Last . . . . .	126
7.8.10	TClassList.Insert . . . . .	126
7.8.11	TClassList.Items . . . . .	126
7.9	TComponentList . . . . .	127
7.9.1	Description . . . . .	127
7.9.2	Method overview . . . . .	127
7.9.3	Property overview . . . . .	127
7.9.4	TComponentList.Destroy . . . . .	127
7.9.5	TComponentList.Add . . . . .	127
7.9.6	TComponentList.Extract . . . . .	128
7.9.7	TComponentList.Remove . . . . .	128
7.9.8	TComponentList.IndexOf . . . . .	128
7.9.9	TComponentList.First . . . . .	129
7.9.10	TComponentList.Last . . . . .	129
7.9.11	TComponentList.Insert . . . . .	129
7.9.12	TComponentList.Items . . . . .	129
7.10	TCustomBucketList . . . . .	130
7.10.1	Description . . . . .	130
7.10.2	Method overview . . . . .	130
7.10.3	Property overview . . . . .	130
7.10.4	TCustomBucketList.Destroy . . . . .	130
7.10.5	TCustomBucketList.Clear . . . . .	130
7.10.6	TCustomBucketList.Add . . . . .	131
7.10.7	TCustomBucketList.Assign . . . . .	131
7.10.8	TCustomBucketList.Exists . . . . .	131

7.10.9	TCustomBucketList.Find . . . . .	131
7.10.10	TCustomBucketList.ForEach . . . . .	132
7.10.11	TCustomBucketList.Remove . . . . .	132
7.10.12	TCustomBucketList.Data . . . . .	132
7.11	TFPCustomHashTable . . . . .	132
7.11.1	Description . . . . .	132
7.11.2	Method overview . . . . .	133
7.11.3	Property overview . . . . .	133
7.11.4	TFPCustomHashTable.Create . . . . .	133
7.11.5	TFPCustomHashTable.CreateWith . . . . .	134
7.11.6	TFPCustomHashTable.Destroy . . . . .	134
7.11.7	TFPCustomHashTable.ChangeTableSize . . . . .	134
7.11.8	TFPCustomHashTable.Clear . . . . .	134
7.11.9	TFPCustomHashTable.Delete . . . . .	135
7.11.10	TFPCustomHashTable.Find . . . . .	135
7.11.11	TFPCustomHashTable.IsEmpty . . . . .	135
7.11.12	TFPCustomHashTable.HashFunction . . . . .	135
7.11.13	TFPCustomHashTable.Count . . . . .	136
7.11.14	TFPCustomHashTable.HashTableSize . . . . .	136
7.11.15	TFPCustomHashTable.HashTable . . . . .	136
7.11.16	TFPCustomHashTable.VoidSlots . . . . .	136
7.11.17	TFPCustomHashTable.LoadFactor . . . . .	137
7.11.18	TFPCustomHashTable.AVGChainLen . . . . .	137
7.11.19	TFPCustomHashTable.MaxChainLength . . . . .	137
7.11.20	TFPCustomHashTable.NumberOfCollisions . . . . .	137
7.11.21	TFPCustomHashTable.Density . . . . .	138
7.12	TFPDataHashTable . . . . .	138
7.12.1	Description . . . . .	138
7.12.2	Method overview . . . . .	138
7.12.3	Property overview . . . . .	138
7.12.4	TFPDataHashTable.Iterate . . . . .	138
7.12.5	TFPDataHashTable.Add . . . . .	139
7.12.6	TFPDataHashTable.Items . . . . .	139
7.13	TFPHashList . . . . .	139
7.13.1	Description . . . . .	139
7.13.2	Method overview . . . . .	140
7.13.3	Property overview . . . . .	140
7.13.4	TFPHashList.Create . . . . .	140
7.13.5	TFPHashList.Destroy . . . . .	140
7.13.6	TFPHashList.Add . . . . .	141

7.13.7	TFPHashList.Clear . . . . .	141
7.13.8	TFPHashList.NameOfIndex . . . . .	141
7.13.9	TFPHashList.HashOfIndex . . . . .	141
7.13.10	TFPHashList.GetNextCollision . . . . .	142
7.13.11	TFPHashList.Delete . . . . .	142
7.13.12	TFPHashList.Error . . . . .	142
7.13.13	TFPHashList.Expand . . . . .	142
7.13.14	TFPHashList.Extract . . . . .	143
7.13.15	TFPHashList.IndexOf . . . . .	143
7.13.16	TFPHashList.Find . . . . .	143
7.13.17	TFPHashList.FindIndexOf . . . . .	143
7.13.18	TFPHashList.FindWithHash . . . . .	144
7.13.19	TFPHashList.Rename . . . . .	144
7.13.20	TFPHashList.Remove . . . . .	144
7.13.21	TFPHashList.Pack . . . . .	144
7.13.22	TFPHashList.ShowStatistics . . . . .	145
7.13.23	TFPHashList.ForEachCall . . . . .	145
7.13.24	TFPHashList.Capacity . . . . .	145
7.13.25	TFPHashList.Count . . . . .	145
7.13.26	TFPHashList.Items . . . . .	146
7.13.27	TFPHashList.List . . . . .	146
7.13.28	TFPHashList.Strs . . . . .	146
7.14	TFPHashObject . . . . .	146
7.14.1	Description . . . . .	146
7.14.2	Method overview . . . . .	147
7.14.3	Property overview . . . . .	147
7.14.4	TFPHashObject.CreateNotOwned . . . . .	147
7.14.5	TFPHashObject.Create . . . . .	147
7.14.6	TFPHashObject.ChangeOwner . . . . .	147
7.14.7	TFPHashObject.ChangeOwnerAndName . . . . .	148
7.14.8	TFPHashObject.Rename . . . . .	148
7.14.9	TFPHashObject.Name . . . . .	148
7.14.10	TFPHashObject.Hash . . . . .	148
7.15	TFPHashObjectList . . . . .	149
7.15.1	Method overview . . . . .	149
7.15.2	Property overview . . . . .	149
7.15.3	TFPHashObjectList.Create . . . . .	149
7.15.4	TFPHashObjectList.Destroy . . . . .	149
7.15.5	TFPHashObjectList.Clear . . . . .	150
7.15.6	TFPHashObjectList.Add . . . . .	150

7.15.7	TFPHashObjectList.NameOfIndex . . . . .	150
7.15.8	TFPHashObjectList.HashOfIndex . . . . .	151
7.15.9	TFPHashObjectList.GetNextCollision . . . . .	151
7.15.10	TFPHashObjectList.Delete . . . . .	151
7.15.11	TFPHashObjectList.Expand . . . . .	151
7.15.12	TFPHashObjectList.Extract . . . . .	152
7.15.13	TFPHashObjectList.Remove . . . . .	152
7.15.14	TFPHashObjectList.IndexOf . . . . .	152
7.15.15	TFPHashObjectList.Find . . . . .	152
7.15.16	TFPHashObjectList.FindIndexOf . . . . .	153
7.15.17	TFPHashObjectList.FindWithHash . . . . .	153
7.15.18	TFPHashObjectList.Rename . . . . .	153
7.15.19	TFPHashObjectList.FindInstanceOf . . . . .	153
7.15.20	TFPHashObjectList.Pack . . . . .	154
7.15.21	TFPHashObjectList.ShowStatistics . . . . .	154
7.15.22	TFPHashObjectList.ForEachCall . . . . .	154
7.15.23	TFPHashObjectList.Capacity . . . . .	154
7.15.24	TFPHashObjectList.Count . . . . .	155
7.15.25	TFPHashObjectList.OwnsObjects . . . . .	155
7.15.26	TFPHashObjectList.Items . . . . .	155
7.15.27	TFPHashObjectList.List . . . . .	155
7.16	TFPObjectHashTable . . . . .	156
7.16.1	Description . . . . .	156
7.16.2	Method overview . . . . .	156
7.16.3	Property overview . . . . .	156
7.16.4	TFPObjectHashTable.Create . . . . .	156
7.16.5	TFPObjectHashTable.CreateWith . . . . .	156
7.16.6	TFPObjectHashTable.Iterate . . . . .	157
7.16.7	TFPObjectHashTable.Add . . . . .	157
7.16.8	TFPObjectHashTable.Items . . . . .	157
7.16.9	TFPObjectHashTable.OwnsObjects . . . . .	158
7.17	TFPOBJECTList . . . . .	158
7.17.1	Description . . . . .	158
7.17.2	Method overview . . . . .	158
7.17.3	Property overview . . . . .	159
7.17.4	TFPOBJECTList.Create . . . . .	159
7.17.5	TFPOBJECTList.Destroy . . . . .	159
7.17.6	TFPOBJECTList.Clear . . . . .	159
7.17.7	TFPOBJECTList.Add . . . . .	159
7.17.8	TFPOBJECTList.Delete . . . . .	160

7.17.9	TFPObjectList.Exchange . . . . .	160
7.17.10	TFPObjectList.Expand . . . . .	160
7.17.11	TFPObjectList.Extract . . . . .	161
7.17.12	TFPObjectList.Remove . . . . .	161
7.17.13	TFPObjectList.IndexOf . . . . .	161
7.17.14	TFPObjectList.FindInstanceOf . . . . .	161
7.17.15	TFPObjectList.Insert . . . . .	162
7.17.16	TFPObjectList.First . . . . .	162
7.17.17	TFPObjectList.Last . . . . .	162
7.17.18	TFPObjectList.Move . . . . .	163
7.17.19	TFPObjectList.Assign . . . . .	163
7.17.20	TFPObjectList.Pack . . . . .	163
7.17.21	TFPObjectList.Sort . . . . .	163
7.17.22	TFPObjectList.ForEachCall . . . . .	164
7.17.23	TFPObjectList.Capacity . . . . .	164
7.17.24	TFPObjectList.Count . . . . .	164
7.17.25	TFPObjectList.OwnsObjects . . . . .	165
7.17.26	TFPObjectList.Items . . . . .	165
7.17.27	TFPObjectList.List . . . . .	165
7.18	TFPStringHashTable . . . . .	165
7.18.1	Description . . . . .	165
7.18.2	Method overview . . . . .	166
7.18.3	Property overview . . . . .	166
7.18.4	TFPStringHashTable.Iterate . . . . .	166
7.18.5	TFPStringHashTable.Add . . . . .	166
7.18.6	TFPStringHashTable.Items . . . . .	166
7.19	THTCustomNode . . . . .	167
7.19.1	Description . . . . .	167
7.19.2	Method overview . . . . .	167
7.19.3	Property overview . . . . .	167
7.19.4	THTCustomNode.CreateWith . . . . .	167
7.19.5	THTCustomNode.HasKey . . . . .	167
7.19.6	THTCustomNode.Key . . . . .	168
7.20	THTDataNode . . . . .	168
7.20.1	Description . . . . .	168
7.20.2	Property overview . . . . .	168
7.20.3	THTDataNode.Data . . . . .	168
7.21	THTObjectNode . . . . .	168
7.21.1	Description . . . . .	168
7.21.2	Property overview . . . . .	169

7.21.3	THTObjectNode.Data	169
7.22	THTOwnedObjectNode	169
7.22.1	Description	169
7.22.2	Method overview	169
7.22.3	THTOwnedObjectNode.Destroy	169
7.23	THTStringNode	169
7.23.1	Description	169
7.23.2	Property overview	170
7.23.3	THTStringNode.Data	170
7.24	TObjectBucketList	170
7.24.1	Description	170
7.24.2	Method overview	170
7.24.3	Property overview	170
7.24.4	TObjectBucketList.Add	170
7.24.5	TObjectBucketList.Remove	171
7.24.6	TObjectBucketList.Data	171
7.25	TObjectList	171
7.25.1	Description	171
7.25.2	Method overview	171
7.25.3	Property overview	172
7.25.4	TObjectList.create	172
7.25.5	TObjectList.Add	172
7.25.6	TObjectList.Extract	172
7.25.7	TObjectList.Remove	173
7.25.8	TObjectList.IndexOf	173
7.25.9	TObjectList.FindInstanceOf	173
7.25.10	TObjectList.Insert	174
7.25.11	TObjectList.First	174
7.25.12	TObjectList.Last	174
7.25.13	TObjectList.OwesObjects	174
7.25.14	TObjectList.Items	175
7.26	TObjectQueue	175
7.26.1	Method overview	175
7.26.2	TObjectQueue.Push	175
7.26.3	TObjectQueue.Pop	175
7.26.4	TObjectQueue.Peek	176
7.27	TObjectStack	176
7.27.1	Description	176
7.27.2	Method overview	176
7.27.3	TObjectStack.Push	176

7.27.4	TObjectStack.Pop	176
7.27.5	TObjectStack.Peek	177
7.28	TOrderedList	177
7.28.1	Description	177
7.28.2	Method overview	177
7.28.3	TOrderedList.Create	177
7.28.4	TOrderedList.Destroy	178
7.28.5	TOrderedList.Count	178
7.28.6	TOrderedList.AtLeast	178
7.28.7	TOrderedList.Push	178
7.28.8	TOrderedList.Pop	179
7.28.9	TOrderedList.Peek	179
7.29	TQueue	179
7.29.1	Description	179
7.30	TStack	179
7.30.1	Description	179
<b>8</b>	<b>Reference for unit 'CustApp'</b>	<b>180</b>
8.1	Used units	180
8.2	Overview	180
8.3	Constants, types and variables	180
8.3.1	Types	180
8.3.2	Variables	181
8.4	TCustomApplication	181
8.4.1	Description	181
8.4.2	Method overview	181
8.4.3	Property overview	182
8.4.4	TCustomApplication.Create	182
8.4.5	TCustomApplication.Destroy	182
8.4.6	TCustomApplication.HandleException	182
8.4.7	TCustomApplication.Initialize	183
8.4.8	TCustomApplication.Run	183
8.4.9	TCustomApplication.ShowException	183
8.4.10	TCustomApplication.Terminate	184
8.4.11	TCustomApplication.FindOptionIndex	184
8.4.12	TCustomApplication.GetOptionValue	184
8.4.13	TCustomApplication.HasOption	185
8.4.14	TCustomApplication.CheckOptions	185
8.4.15	TCustomApplication.GetEnvironmentList	186
8.4.16	TCustomApplication.Log	186

8.4.17	TCustomApplication.ExeName . . . . .	187
8.4.18	TCustomApplication.HelpFile . . . . .	187
8.4.19	TCustomApplication.Terminated . . . . .	187
8.4.20	TCustomApplication.Title . . . . .	187
8.4.21	TCustomApplication.OnException . . . . .	188
8.4.22	TCustomApplication.ConsoleApplication . . . . .	188
8.4.23	TCustomApplication.Location . . . . .	188
8.4.24	TCustomApplication.Params . . . . .	189
8.4.25	TCustomApplication.ParamCount . . . . .	189
8.4.26	TCustomApplication.EnvironmentVariable . . . . .	189
8.4.27	TCustomApplication.OptionChar . . . . .	189
8.4.28	TCustomApplication.CaseSensitiveOptions . . . . .	190
8.4.29	TCustomApplication.StopOnException . . . . .	190
8.4.30	TCustomApplication.EventLogFilter . . . . .	190
<b>9</b>	<b>Reference for unit 'daemonapp'</b>	<b>191</b>
9.1	Used units . . . . .	191
9.2	Overview . . . . .	191
9.3	Daemon application architecture . . . . .	192
9.4	Constants, types and variables . . . . .	192
9.4.1	Resource strings . . . . .	192
9.4.2	Types . . . . .	193
9.4.3	Variables . . . . .	196
9.5	Procedures and functions . . . . .	196
9.5.1	Application . . . . .	196
9.5.2	DaemonError . . . . .	197
9.5.3	RegisterDaemonApplicationClass . . . . .	197
9.5.4	RegisterDaemonClass . . . . .	197
9.5.5	RegisterDaemonMapper . . . . .	197
9.6	EDaemon . . . . .	198
9.6.1	Description . . . . .	198
9.7	TCustomDaemon . . . . .	198
9.7.1	Description . . . . .	198
9.7.2	Method overview . . . . .	198
9.7.3	Property overview . . . . .	198
9.7.4	TCustomDaemon.LogMessage . . . . .	198
9.7.5	TCustomDaemon.ReportStatus . . . . .	199
9.7.6	TCustomDaemon.Definition . . . . .	199
9.7.7	TCustomDaemon.DaemonThread . . . . .	199
9.7.8	TCustomDaemon.Controller . . . . .	200

9.7.9	TCustomDaemon.Status . . . . .	200
9.7.10	TCustomDaemon.Logger . . . . .	200
9.8	TCustomDaemonApplication . . . . .	200
9.8.1	Description . . . . .	200
9.8.2	Method overview . . . . .	201
9.8.3	Property overview . . . . .	201
9.8.4	TCustomDaemonApplication.Create . . . . .	201
9.8.5	TCustomDaemonApplication.Destroy . . . . .	201
9.8.6	TCustomDaemonApplication.ShowException . . . . .	201
9.8.7	TCustomDaemonApplication.CreateDaemon . . . . .	202
9.8.8	TCustomDaemonApplication.StopDaemons . . . . .	202
9.8.9	TCustomDaemonApplication.InstallDaemons . . . . .	202
9.8.10	TCustomDaemonApplication.RunDaemons . . . . .	202
9.8.11	TCustomDaemonApplication.UnInstallDaemons . . . . .	203
9.8.12	TCustomDaemonApplication.ShowHelp . . . . .	203
9.8.13	TCustomDaemonApplication.CreateForm . . . . .	203
9.8.14	TCustomDaemonApplication.OnRun . . . . .	203
9.8.15	TCustomDaemonApplication.EventLog . . . . .	204
9.8.16	TCustomDaemonApplication.GUIMainLoop . . . . .	204
9.8.17	TCustomDaemonApplication.GuiHandle . . . . .	204
9.8.18	TCustomDaemonApplication.RunMode . . . . .	204
9.8.19	TCustomDaemonApplication.AutoRegisterMessageFile . . . . .	205
9.9	TCustomDaemonMapper . . . . .	205
9.9.1	Description . . . . .	205
9.9.2	Method overview . . . . .	205
9.9.3	Property overview . . . . .	205
9.9.4	TCustomDaemonMapper.Create . . . . .	205
9.9.5	TCustomDaemonMapper.Destroy . . . . .	206
9.9.6	TCustomDaemonMapper.DaemonDefs . . . . .	206
9.9.7	TCustomDaemonMapper.OnCreate . . . . .	206
9.9.8	TCustomDaemonMapper.OnDestroy . . . . .	206
9.9.9	TCustomDaemonMapper.OnRun . . . . .	207
9.9.10	TCustomDaemonMapper.OnInstall . . . . .	207
9.9.11	TCustomDaemonMapper.OnUnInstall . . . . .	207
9.10	TDaemon . . . . .	207
9.10.1	Description . . . . .	207
9.10.2	Property overview . . . . .	208
9.10.3	TDaemon.Definition . . . . .	208
9.10.4	TDaemon.Status . . . . .	208
9.10.5	TDaemon.OnStart . . . . .	208

9.10.6	TDaemon.OnStop	209
9.10.7	TDaemon.OnPause	209
9.10.8	TDaemon.OnContinue	209
9.10.9	TDaemon.OnShutDown	210
9.10.10	TDaemon.OnExecute	210
9.10.11	TDaemon.BeforeInstall	210
9.10.12	TDaemon.AfterInstall	211
9.10.13	TDaemon.BeforeUnInstall	211
9.10.14	TDaemon.AfterUnInstall	211
9.10.15	TDaemon.OnControlCode	211
9.11	TDaemonApplication	212
9.11.1	Description	212
9.12	TDaemonController	212
9.12.1	Description	212
9.12.2	Method overview	212
9.12.3	Property overview	212
9.12.4	TDaemonController.Create	212
9.12.5	TDaemonController.Destroy	213
9.12.6	TDaemonController.StartService	213
9.12.7	TDaemonController.Main	213
9.12.8	TDaemonController.Controller	213
9.12.9	TDaemonController.ReportStatus	214
9.12.10	TDaemonController.Daemon	214
9.12.11	TDaemonController.Params	214
9.12.12	TDaemonController.LastStatus	214
9.12.13	TDaemonController.CheckPoint	215
9.13	TDaemonDef	215
9.13.1	Description	215
9.13.2	Method overview	215
9.13.3	Property overview	215
9.13.4	TDaemonDef.Create	215
9.13.5	TDaemonDef.Destroy	216
9.13.6	TDaemonDef.DaemonClass	216
9.13.7	TDaemonDef.Instance	216
9.13.8	TDaemonDef.DaemonClassName	216
9.13.9	TDaemonDef.Name	217
9.13.10	TDaemonDef.Description	217
9.13.11	TDaemonDef.DisplayName	217
9.13.12	TDaemonDef.RunArguments	217
9.13.13	TDaemonDef.Options	218

9.13.14 TDaemonDef.Enabled . . . . .	218
9.13.15 TDaemonDef.WinBindings . . . . .	218
9.13.16 TDaemonDef.OnCreateInstance . . . . .	218
9.13.17 TDaemonDef.LogStatusReport . . . . .	219
9.14 TDaemonDefs . . . . .	219
9.14.1 Description . . . . .	219
9.14.2 Method overview . . . . .	219
9.14.3 Property overview . . . . .	219
9.14.4 TDaemonDefs.Create . . . . .	219
9.14.5 TDaemonDefs.IndexOfDaemonDef . . . . .	220
9.14.6 TDaemonDefs.FindDaemonDef . . . . .	220
9.14.7 TDaemonDefs.DaemonDefByName . . . . .	220
9.14.8 TDaemonDefs.Daemons . . . . .	220
9.15 TDaemonMapper . . . . .	221
9.15.1 Description . . . . .	221
9.15.2 Method overview . . . . .	221
9.15.3 TDaemonMapper.Create . . . . .	221
9.15.4 TDaemonMapper.CreateNew . . . . .	221
9.16 TDaemonThread . . . . .	222
9.16.1 Description . . . . .	222
9.16.2 Method overview . . . . .	222
9.16.3 Property overview . . . . .	222
9.16.4 TDaemonThread.Create . . . . .	222
9.16.5 TDaemonThread.Execute . . . . .	222
9.16.6 TDaemonThread.CheckControlMessage . . . . .	223
9.16.7 TDaemonThread.StopDaemon . . . . .	223
9.16.8 TDaemonThread.PauseDaemon . . . . .	223
9.16.9 TDaemonThread.ContinueDaemon . . . . .	223
9.16.10 TDaemonThread.ShutDownDaemon . . . . .	224
9.16.11 TDaemonThread.InterrogateDaemon . . . . .	224
9.16.12 TDaemonThread.Daemon . . . . .	224
9.17 TDependencies . . . . .	224
9.17.1 Description . . . . .	224
9.17.2 Method overview . . . . .	224
9.17.3 Property overview . . . . .	225
9.17.4 TDependencies.Create . . . . .	225
9.17.5 TDependencies.Items . . . . .	225
9.18 TDependency . . . . .	225
9.18.1 Description . . . . .	225
9.18.2 Method overview . . . . .	225

9.18.3	Property overview . . . . .	225
9.18.4	TDependency.Assign . . . . .	226
9.18.5	TDependency.Name . . . . .	226
9.18.6	TDependency.IsGroup . . . . .	226
9.19	TWinBindings . . . . .	226
9.19.1	Description . . . . .	226
9.19.2	Method overview . . . . .	226
9.19.3	Property overview . . . . .	227
9.19.4	TWinBindings.Create . . . . .	227
9.19.5	TWinBindings.Destroy . . . . .	227
9.19.6	TWinBindings.Assign . . . . .	227
9.19.7	TWinBindings.ErrCode . . . . .	227
9.19.8	TWinBindings.Win32ErrCode . . . . .	228
9.19.9	TWinBindings.Dependencies . . . . .	228
9.19.10	TWinBindings.GroupName . . . . .	228
9.19.11	TWinBindings.Password . . . . .	229
9.19.12	TWinBindings.UserName . . . . .	229
9.19.13	TWinBindings.StartType . . . . .	229
9.19.14	TWinBindings.WaitHint . . . . .	229
9.19.15	TWinBindings.IDTag . . . . .	230
9.19.16	TWinBindings.ServiceType . . . . .	230
9.19.17	TWinBindings.ErrorSeverity . . . . .	230
<b>10</b>	<b>Reference for unit 'db'</b>	<b>231</b>
10.1	Used units . . . . .	231
10.2	Overview . . . . .	231
10.3	Constants, types and variables . . . . .	231
10.3.1	Constants . . . . .	231
10.3.2	Types . . . . .	232
10.4	Procedures and functions . . . . .	246
10.4.1	BuffersEqual . . . . .	246
10.4.2	DatabaseError . . . . .	246
10.4.3	DatabaseErrorFmt . . . . .	246
10.4.4	DateTimeRecToDateTime . . . . .	246
10.4.5	DateTimeToDateTimeRec . . . . .	247
10.4.6	DisposeMem . . . . .	247
10.4.7	ExtractFieldName . . . . .	247
10.4.8	SkipComments . . . . .	248
10.5	EDatabaseError . . . . .	248
10.5.1	Description . . . . .	248

10.6	EUpdateError	248
10.6.1	Description	248
10.6.2	Method overview	248
10.6.3	Property overview	248
10.6.4	EUpdateError.Create	249
10.6.5	EUpdateError.Destroy	249
10.6.6	EUpdateError.Context	249
10.6.7	EUpdateError.ErrorCode	249
10.6.8	EUpdateError.OriginalException	250
10.6.9	EUpdateError.PreviousError	250
10.7	IParserSupport	250
10.7.1	Description	250
10.7.2	Method overview	251
10.7.3	IParserSupport.PSEndTransaction	251
10.7.4	IParserSupport.PSExecute	251
10.7.5	IParserSupport.PSExecuteStatement	251
10.7.6	IParserSupport.PSGetAttributes	252
10.7.7	IParserSupport.PSGetCommandText	252
10.7.8	IParserSupport.PSGetCommandType	252
10.7.9	IParserSupport.PSGetDefaultOrder	253
10.7.10	IParserSupport.PSGetIndexDefs	253
10.7.11	IParserSupport.PSGetKeyFields	253
10.7.12	IParserSupport.PSGetParams	253
10.7.13	IParserSupport.PSGetQuoteChar	254
10.7.14	IParserSupport.PSGetTableName	254
10.7.15	IParserSupport.PSGetUpdateException	254
10.7.16	IParserSupport.PSInTransaction	254
10.7.17	IParserSupport.PSIsSQLBased	255
10.7.18	IParserSupport.PSIsSQLSupported	255
10.7.19	IParserSupport.PSReset	255
10.7.20	IParserSupport.PSSetCommandText	255
10.7.21	IParserSupport.PSSetParams	256
10.7.22	IParserSupport.PSSStartTransaction	256
10.7.23	IParserSupport.PSUpdateRecord	256
10.8	TAutoIncField	256
10.8.1	Description	256
10.8.2	Method overview	257
10.8.3	TAutoIncField.Create	257
10.9	TBCDField	257
10.9.1	Description	257

10.9.2 Method overview . . . . .	257
10.9.3 Property overview . . . . .	257
10.9.4 TBCDField.Create . . . . .	257
10.9.5 TBCDField.CheckRange . . . . .	258
10.9.6 TBCDField.Value . . . . .	258
10.9.7 TBCDField.Precision . . . . .	258
10.9.8 TBCDField.Currency . . . . .	259
10.9.9 TBCDField.MaxValue . . . . .	259
10.9.10 TBCDField.MinValue . . . . .	259
10.9.11 TBCDField.Size . . . . .	260
10.10 TBinaryField . . . . .	260
10.10.1 Description . . . . .	260
10.10.2 Method overview . . . . .	260
10.10.3 Property overview . . . . .	260
10.10.4 TBinaryField.Create . . . . .	260
10.10.5 TBinaryField.Size . . . . .	261
10.11 TBlobField . . . . .	261
10.11.1 Description . . . . .	261
10.11.2 Method overview . . . . .	261
10.11.3 Property overview . . . . .	261
10.11.4 TBlobField.Create . . . . .	261
10.11.5 TBlobField.Clear . . . . .	262
10.11.6 TBlobField.IsBlob . . . . .	262
10.11.7 TBlobField.LoadFromFile . . . . .	262
10.11.8 TBlobField.LoadFromStream . . . . .	262
10.11.9 TBlobField.SaveToFile . . . . .	263
10.11.10 TBlobField.SaveToStream . . . . .	263
10.11.11 TBlobField.SetFieldType . . . . .	263
10.11.12 TBlobField.BlobSize . . . . .	263
10.11.13 TBlobField.Modified . . . . .	264
10.11.14 TBlobField.Value . . . . .	264
10.11.15 TBlobField.Transliterate . . . . .	264
10.11.16 TBlobField.BlobType . . . . .	265
10.11.17 TBlobField.Size . . . . .	265
10.12 TBooleanField . . . . .	265
10.12.1 Description . . . . .	265
10.12.2 Method overview . . . . .	265
10.12.3 Property overview . . . . .	265
10.12.4 TBooleanField.Create . . . . .	266
10.12.5 TBooleanField.Value . . . . .	266

10.12.6 TBooleanField.DisplayValues . . . . .	266
10.13 TBytesField . . . . .	266
10.13.1 Description . . . . .	266
10.13.2 Method overview . . . . .	267
10.13.3 TBytesField.Create . . . . .	267
10.14 TCheckConstraint . . . . .	267
10.14.1 Description . . . . .	267
10.14.2 Method overview . . . . .	267
10.14.3 Property overview . . . . .	267
10.14.4 TCheckConstraint.Assign . . . . .	268
10.14.5 TCheckConstraint.CustomConstraint . . . . .	268
10.14.6 TCheckConstraint.ErrorMessage . . . . .	268
10.14.7 TCheckConstraint.FromDictionary . . . . .	268
10.14.8 TCheckConstraint.ImportedConstraint . . . . .	269
10.15 TCheckConstraints . . . . .	269
10.15.1 Description . . . . .	269
10.15.2 Method overview . . . . .	269
10.15.3 Property overview . . . . .	269
10.15.4 TCheckConstraints.Create . . . . .	269
10.15.5 TCheckConstraints.Add . . . . .	270
10.15.6 TCheckConstraints.Items . . . . .	270
10.16 TCurrencyField . . . . .	270
10.16.1 Description . . . . .	270
10.16.2 Method overview . . . . .	270
10.16.3 Property overview . . . . .	270
10.16.4 TCurrencyField.Create . . . . .	271
10.16.5 TCurrencyField.Currency . . . . .	271
10.17 TCustomConnection . . . . .	271
10.17.1 Description . . . . .	271
10.17.2 Method overview . . . . .	271
10.17.3 Property overview . . . . .	272
10.17.4 TCustomConnection.Close . . . . .	272
10.17.5 TCustomConnection.Destroy . . . . .	272
10.17.6 TCustomConnection.Open . . . . .	272
10.17.7 TCustomConnection.DataSetCount . . . . .	273
10.17.8 TCustomConnection.DataSets . . . . .	273
10.17.9 TCustomConnection.Connected . . . . .	273
10.17.10 TCustomConnection.LoginPrompt . . . . .	274
10.17.11 ICustomConnection.AfterConnect . . . . .	274
10.17.12 TCustomConnection.AfterDisconnect . . . . .	274

10.17.13 <code>ICustomConnection.BeforeConnect</code>	275
10.17.14 <code>ICustomConnection.BeforeDisconnect</code>	275
10.17.15 <code>ICustomConnection.OnLogin</code>	275
10.18 <code>TDatabase</code>	276
10.18.1 <code>Description</code>	276
10.18.2 <code>Method overview</code>	276
10.18.3 <code>Property overview</code>	276
10.18.4 <code>TDatabase.Create</code>	276
10.18.5 <code>TDatabase.Destroy</code>	277
10.18.6 <code>TDatabase.CloseDataSets</code>	277
10.18.7 <code>TDatabase.CloseTransactions</code>	277
10.18.8 <code>TDatabase.StartTransaction</code>	277
10.18.9 <code>TDatabase.EndTransaction</code>	278
10.18.10 <code>TDatabase.TransactionCount</code>	278
10.18.11 <code>TDatabase.Transactions</code>	278
10.18.12 <code>TDatabase.Directory</code>	278
10.18.13 <code>TDatabase.IsSQLBased</code>	279
10.18.14 <code>TDatabase.Connected</code>	279
10.18.15 <code>TDatabase.DatabaseName</code>	279
10.18.16 <code>TDatabase.KeepConnection</code>	279
10.18.17 <code>TDatabase.Params</code>	280
10.19 <code>TDataLink</code>	280
10.19.1 <code>Description</code>	280
10.19.2 <code>Method overview</code>	280
10.19.3 <code>Property overview</code>	281
10.19.4 <code>TDataLink.Create</code>	281
10.19.5 <code>TDataLink.Destroy</code>	281
10.19.6 <code>TDataLink.Edit</code>	281
10.19.7 <code>TDataLink.UpdateRecord</code>	282
10.19.8 <code>TDataLink.ExecuteAction</code>	282
10.19.9 <code>TDataLink.UpdateAction</code>	282
10.19.10 <code>TDataLink.Active</code>	282
10.19.11 <code>TDataLink.ActiveRecord</code>	283
10.19.12 <code>TDataLink.BOF</code>	283
10.19.13 <code>TDataLink.BufferCount</code>	283
10.19.14 <code>TDataLink.DataSet</code>	284
10.19.15 <code>TDataLink.DataSource</code>	284
10.19.16 <code>TDataLink.DataSourceFixed</code>	284
10.19.17 <code>TDataLink.Editing</code>	284
10.19.18 <code>TDataLink.Eof</code>	285

10.19.19TDataLink.ReadOnly . . . . .	285
10.19.20TDataLink.RecordCount . . . . .	285
10.20TDataSet . . . . .	285
10.20.1 Description . . . . .	285
10.20.2 Method overview . . . . .	288
10.20.3 Property overview . . . . .	289
10.20.4 TDataSet.Create . . . . .	290
10.20.5 TDataSet.Destroy . . . . .	290
10.20.6 TDataSet.ActiveBuffer . . . . .	290
10.20.7 TDataSet.GetFieldData . . . . .	290
10.20.8 TDataSet.SetFieldData . . . . .	291
10.20.9 TDataSet.Append . . . . .	291
10.20.10TDataSet.AppendRecord . . . . .	291
10.20.11TDataSet.BookmarkValid . . . . .	292
10.20.12TDataSet.Cancel . . . . .	292
10.20.13TDataSet.CheckBrowseMode . . . . .	292
10.20.14TDataSet.ClearFields . . . . .	292
10.20.15TDataSet.Close . . . . .	293
10.20.16TDataSet.ControlsDisabled . . . . .	293
10.20.17TDataSet.CompareBookmarks . . . . .	293
10.20.18TDataSet.CreateBlobStream . . . . .	294
10.20.19TDataSet.CursorPosChanged . . . . .	294
10.20.20TDataSet.DataConvert . . . . .	294
10.20.21TDataSet.Delete . . . . .	294
10.20.22TDataSet.DisableControls . . . . .	295
10.20.23TDataSet.Edit . . . . .	295
10.20.24TDataSet.EnableControls . . . . .	296
10.20.25TDataSet.FieldName . . . . .	296
10.20.26TDataSet.FindField . . . . .	296
10.20.27TDataSet.FindFirst . . . . .	297
10.20.28TDataSet.FindLast . . . . .	297
10.20.29TDataSet.FindNext . . . . .	297
10.20.30TDataSet.FindPrior . . . . .	297
10.20.31TDataSet.First . . . . .	298
10.20.32TDataSet.FreeBookmark . . . . .	298
10.20.33TDataSet.GetBookmark . . . . .	298
10.20.34TDataSet.GetCurrentRecord . . . . .	299
10.20.35TDataSet.GetFieldList . . . . .	299
10.20.36TDataSet.GetFieldNames . . . . .	299
10.20.37TDataSet.GotoBookmark . . . . .	299

10.20.3 <code>TDataSet.Insert</code>	300
10.20.3 <code>9TDataSet.InsertRecord</code>	300
10.20.4 <code>0TDataSet.IsEmpty</code>	300
10.20.4 <code>1TDataSet.IsLinkedTo</code>	300
10.20.4 <code>2TDataSet.IsSequenced</code>	301
10.20.4 <code>3TDataSet.Last</code>	301
10.20.4 <code>4TDataSet.Locate</code>	301
10.20.4 <code>5TDataSet.Lookup</code>	302
10.20.4 <code>6TDataSet.MoveBy</code>	302
10.20.4 <code>7TDataSet.Next</code>	302
10.20.4 <code>8TDataSet.Open</code>	303
10.20.4 <code>9TDataSet.Post</code>	303
10.20.5 <code>0TDataSet.Prior</code>	304
10.20.5 <code>1TDataSet.Refresh</code>	304
10.20.5 <code>2TDataSet.Resync</code>	304
10.20.5 <code>3TDataSet.SetFields</code>	304
10.20.5 <code>4TDataSet.Translate</code>	305
10.20.5 <code>5TDataSet.UpdateCursorPos</code>	305
10.20.5 <code>6TDataSet.UpdateRecord</code>	305
10.20.5 <code>7TDataSet.UpdateStatus</code>	306
10.20.5 <code>8TDataSet.BlockReadSize</code>	306
10.20.5 <code>9TDataSet.BOF</code>	306
10.20.6 <code>0TDataSet.Bookmark</code>	306
10.20.6 <code>1TDataSet.CanModify</code>	307
10.20.6 <code>2TDataSet.DataSource</code>	308
10.20.6 <code>3TDataSet.DefaultFields</code>	308
10.20.6 <code>4TDataSet.EOF</code>	308
10.20.6 <code>5TDataSet.FieldCount</code>	309
10.20.6 <code>6TDataSet.FieldDefs</code>	309
10.20.6 <code>7TDataSet.Found</code>	310
10.20.6 <code>8TDataSet.Modified</code>	310
10.20.6 <code>9TDataSet.IsUniDirectional</code>	310
10.20.7 <code>0TDataSet.RecordCount</code>	311
10.20.7 <code>1TDataSet.RecNo</code>	311
10.20.7 <code>2TDataSet.RecordSize</code>	311
10.20.7 <code>3TDataSet.State</code>	312
10.20.7 <code>4TDataSet.Fields</code>	312
10.20.7 <code>5TDataSet.FieldValues</code>	312
10.20.7 <code>6TDataSet.Filter</code>	313
10.20.7 <code>7TDataSet.Filtered</code>	313

10.20.7 <code>TDataSet.FilterOptions</code>	313
10.20.79 <code>TDataSet.Active</code>	314
10.20.80 <code>TDataSet.AutoCalcFields</code>	314
10.20.81 <code>ITDataSet.BeforeOpen</code>	314
10.20.82 <code>TDataset.AfterOpen</code>	315
10.20.83 <code>TDataSet.BeforeClose</code>	315
10.20.84 <code>TDataSet.AfterClose</code>	315
10.20.85 <code>TDataSet.BeforeInsert</code>	315
10.20.86 <code>TDataSet.AfterInsert</code>	316
10.20.87 <code>TDataSet.BeforeEdit</code>	316
10.20.88 <code>TDataSet.AfterEdit</code>	316
10.20.89 <code>TDataSet.BeforePost</code>	317
10.20.90 <code>ITDataSet.AfterPost</code>	317
10.20.91 <code>ITDataSet.BeforeCancel</code>	317
10.20.92 <code>TDataset.AfterCancel</code>	318
10.20.93 <code>TDataSet.BeforeDelete</code>	318
10.20.94 <code>TDataSet.AfterDelete</code>	318
10.20.95 <code>TDataSet.BeforeScroll</code>	318
10.20.96 <code>TDataSet.AfterScroll</code>	319
10.20.97 <code>TDataSet.BeforeRefresh</code>	319
10.20.98 <code>TDataSet.AfterRefresh</code>	319
10.20.99 <code>TDataSet.OnCalcFields</code>	320
10.20.100 <code>IDDataSet.OnDeleteError</code>	320
10.20.101 <code>IDDataSet.OnEditError</code>	321
10.20.102 <code>IDDataSet.OnFilterRecord</code>	321
10.20.103 <code>IDDataSet.OnNewRecord</code>	321
10.20.104 <code>IDDataSet.OnPostError</code>	322
10.21 <code>TDataSource</code>	322
10.21.1 <code>Description</code>	322
10.21.2 <code>Method overview</code>	322
10.21.3 <code>Property overview</code>	323
10.21.4 <code>TDataSource.Create</code>	323
10.21.5 <code>TDataSource.Destroy</code>	323
10.21.6 <code>TDataSource.Edit</code>	323
10.21.7 <code>TDataSource.IsLinkedTo</code>	324
10.21.8 <code>TDataSource.State</code>	324
10.21.9 <code>TDataSource.AutoEdit</code>	324
10.21.10 <code>TDataSource.DataSet</code>	324
10.21.11 <code>ITDataSource.Enabled</code>	325
10.21.12 <code>TDatasource.OnStateChange</code>	325

10.21.13 TDataSource.OnDataChange . . . . .	325
10.21.14 TDataSource.OnUpdateData . . . . .	326
10.22 TDateField . . . . .	326
10.22.1 Description . . . . .	326
10.22.2 Method overview . . . . .	326
10.22.3 TDateField.Create . . . . .	326
10.23 TDatetimeField . . . . .	326
10.23.1 Description . . . . .	326
10.23.2 Method overview . . . . .	327
10.23.3 Property overview . . . . .	327
10.23.4 TDatetimeField.Create . . . . .	327
10.23.5 TDatetimeField.Value . . . . .	327
10.23.6 TDatetimeField.DisplayFormat . . . . .	327
10.23.7 TDatetimeField.EditMask . . . . .	328
10.24 TDBDataset . . . . .	328
10.24.1 Description . . . . .	328
10.24.2 Method overview . . . . .	328
10.24.3 Property overview . . . . .	328
10.24.4 TDBDataset.destroy . . . . .	329
10.24.5 TDBDataset.DataBase . . . . .	329
10.24.6 TDBDataset.Transaction . . . . .	329
10.25 TDBTransaction . . . . .	329
10.25.1 Description . . . . .	329
10.25.2 Method overview . . . . .	330
10.25.3 Property overview . . . . .	330
10.25.4 TDBTransaction.Create . . . . .	330
10.25.5 TDBTransaction.destroy . . . . .	330
10.25.6 TDBTransaction.CloseDataSets . . . . .	330
10.25.7 TDBTransaction.DataBase . . . . .	331
10.25.8 TDBTransaction.Active . . . . .	331
10.26 TDefCollection . . . . .	331
10.26.1 Description . . . . .	331
10.26.2 Method overview . . . . .	331
10.26.3 Property overview . . . . .	331
10.26.4 TDefCollection.create . . . . .	332
10.26.5 TDefCollection.Find . . . . .	332
10.26.6 TDefCollection.GetItemNames . . . . .	332
10.26.7 TDefCollection.IndexOf . . . . .	332
10.26.8 TDefCollection.Dataset . . . . .	333
10.26.9 TDefCollection.Updated . . . . .	333

10.27 TDetailDataLink . . . . .	333
10.27.1 Description . . . . .	333
10.27.2 Property overview . . . . .	333
10.27.3 TDetailDataLink.DetailDataSet . . . . .	333
10.28 TField . . . . .	334
10.28.1 Description . . . . .	334
10.28.2 Method overview . . . . .	334
10.28.3 Property overview . . . . .	336
10.28.4 TField.Create . . . . .	337
10.28.5 TField.Destroy . . . . .	337
10.28.6 TField.Assign . . . . .	337
10.28.7 TField.AssignValue . . . . .	337
10.28.8 TField.Clear . . . . .	338
10.28.9 TField.FocusControl . . . . .	338
10.28.10 TField.GetData . . . . .	338
10.28.11 TField.IsBlob . . . . .	339
10.28.12 TField.IsValidChar . . . . .	339
10.28.13 TField.RefreshLookupList . . . . .	339
10.28.14 TField.SetData . . . . .	339
10.28.15 TField.SetFieldType . . . . .	340
10.28.16 TField.Validate . . . . .	340
10.28.17 TField.AsBCD . . . . .	340
10.28.18 TField.AsBoolean . . . . .	341
10.28.19 TField.AsBytes . . . . .	341
10.28.20 TField.AsCurrency . . . . .	341
10.28.21 TField.AsDateTime . . . . .	342
10.28.22 TField.AsFloat . . . . .	342
10.28.23 TField.AsLongint . . . . .	342
10.28.24 TField.AsLargeInt . . . . .	343
10.28.25 TField.AsInteger . . . . .	343
10.28.26 TFieldAsString . . . . .	343
10.28.27 TField.AsWideString . . . . .	344
10.28.28 TField.AsVariant . . . . .	344
10.28.29 TField.AttributeSet . . . . .	344
10.28.30 TField.Calculated . . . . .	345
10.28.31 TField.CanModify . . . . .	345
10.28.32 TField.CurValue . . . . .	345
10.28.33 TField.DataSet . . . . .	345
10.28.34 TField.DataSize . . . . .	346
10.28.35 TField.DataType . . . . .	346

10.28.3 <del>T</del> Field.DisplayName . . . . .	346
10.28.3 <del>T</del> Field.DisplayText . . . . .	346
10.28.3 <del>T</del> Field.EditMask . . . . .	347
10.28.3 <del>T</del> Field.EditMaskPtr . . . . .	347
10.28.4 <del>T</del> Field.FieldNo . . . . .	347
10.28.4 <del>I</del> Field.IsIndexField . . . . .	348
10.28.4 <del>T</del> Field.IsNull . . . . .	348
10.28.4 <del>T</del> Field.Lookup . . . . .	348
10.28.4 <del>T</del> Field.NewValue . . . . .	348
10.28.4 <del>T</del> Field.Offset . . . . .	349
10.28.4 <del>T</del> Field.Size . . . . .	349
10.28.4 <del>T</del> Field.Text . . . . .	349
10.28.4 <del>T</del> Field.ValidChars . . . . .	350
10.28.4 <del>T</del> Field.Value . . . . .	350
10.28.5 <del>T</del> Field.OldValue . . . . .	350
10.28.5 <del>I</del> Field.LookupList . . . . .	351
10.28.5 <del>T</del> Field.Alignment . . . . .	351
10.28.5 <del>T</del> Field.CustomConstraint . . . . .	351
10.28.5 <del>T</del> Field.ConstraintErrorMessage . . . . .	352
10.28.5 <del>T</del> Field.DefaultExpression . . . . .	352
10.28.5 <del>T</del> Field.DisplayLabel . . . . .	352
10.28.5 <del>T</del> Field.DisplayWidth . . . . .	352
10.28.5 <del>T</del> Field.FieldKind . . . . .	353
10.28.5 <del>T</del> Field.FieldName . . . . .	353
10.28.6 <del>T</del> Field.HasConstraints . . . . .	353
10.28.6 <del>I</del> Field.Index . . . . .	354
10.28.6 <del>T</del> Field.ImportedConstraint . . . . .	354
10.28.6 <del>T</del> Field.KeyFields . . . . .	354
10.28.6 <del>T</del> Field.LookupCache . . . . .	354
10.28.6 <del>T</del> Field.LookupDataSet . . . . .	355
10.28.6 <del>T</del> Field.LookupKeyFields . . . . .	355
10.28.6 <del>T</del> Field.LookupResultField . . . . .	355
10.28.6 <del>T</del> Field-Origin . . . . .	356
10.28.6 <del>T</del> Field.ProviderFlags . . . . .	356
10.28.7 <del>T</del> Field.ReadOnly . . . . .	356
10.28.7 <del>I</del> Field.Required . . . . .	357
10.28.7 <del>T</del> Field.Visible . . . . .	357
10.28.7 <del>T</del> Field.OnChange . . . . .	357
10.28.7 <del>T</del> Field.OnGetText . . . . .	358
10.28.7 <del>T</del> Field.OnSetText . . . . .	358

10.28.7 <code>TField.OnValidate</code>	358
10.29 <code>TFieldDef</code>	358
10.29.1 <code>Description</code>	358
10.29.2 <code>Method overview</code>	359
10.29.3 <code>Property overview</code>	359
10.29.4 <code>TFieldDef.Create</code>	359
10.29.5 <code>TFieldDef.Destroy</code>	359
10.29.6 <code>TFieldDef.Assign</code>	360
10.29.7 <code>TFieldDef.CreateField</code>	360
10.29.8 <code>TFieldDef.FieldClass</code>	360
10.29.9 <code>TFieldDef.FieldNo</code>	360
10.29.10 <code>TFieldDef.InternalCalcField</code>	361
10.29.11 <code>ITFieldDef.Required</code>	361
10.29.12 <code>TFieldDef.Attributes</code>	361
10.29.13 <code>ITFieldDef.DataType</code>	361
10.29.14 <code>ITFieldDef.Precision</code>	362
10.29.15 <code>ITFieldDef.Size</code>	362
10.30 <code>TFieldDefs</code>	362
10.30.1 <code>Description</code>	362
10.30.2 <code>Method overview</code>	363
10.30.3 <code>Property overview</code>	363
10.30.4 <code>TFieldDefs.Create</code>	363
10.30.5 <code>TFieldDefs.Add</code>	363
10.30.6 <code>TFieldDefs.AddFieldDef</code>	363
10.30.7 <code>TFieldDefs.Assign</code>	364
10.30.8 <code>TFieldDefs.Find</code>	364
10.30.9 <code>TFieldDefs.Update</code>	364
10.30.10 <code>ITFieldDefs.MakeNameUnique</code>	364
10.30.11 <code>ITFieldDefs.HiddenFields</code>	365
10.30.12 <code>ITFieldDefs.Items</code>	365
10.31 <code>TFields</code>	365
10.31.1 <code>Description</code>	365
10.31.2 <code>Method overview</code>	365
10.31.3 <code>Property overview</code>	366
10.31.4 <code>TFields.Create</code>	366
10.31.5 <code>TFields.Destroy</code>	366
10.31.6 <code>TFields.Add</code>	366
10.31.7 <code>TFields.CheckFieldName</code>	366
10.31.8 <code>TFields.CheckFieldNames</code>	367
10.31.9 <code>TFields.Clear</code>	367

10.31.10 TFields.FindField . . . . .	367
10.31.11 IFields.FieldName . . . . .	367
10.31.12 TFields.FieldByNumber . . . . .	368
10.31.13 TFields.GetEnumerator . . . . .	368
10.31.14 TFields.GetFieldNames . . . . .	368
10.31.15 TFields.IndexOf . . . . .	368
10.31.16 TFields.Remove . . . . .	369
10.31.17 TFields.Count . . . . .	369
10.31.18 TFields.Dataset . . . . .	369
10.31.19 TFields.Fields . . . . .	369
10.32 TFieldsEnumerator . . . . .	370
10.32.1 Description . . . . .	370
10.32.2 Method overview . . . . .	370
10.32.3 Property overview . . . . .	370
10.32.4 TFieldsEnumerator.Create . . . . .	370
10.32.5 TFieldsEnumerator.MoveNext . . . . .	370
10.32.6 TFieldsEnumerator.Current . . . . .	371
10.33 TFloatField . . . . .	371
10.33.1 Description . . . . .	371
10.33.2 Method overview . . . . .	371
10.33.3 Property overview . . . . .	371
10.33.4 TFfloatField.Create . . . . .	371
10.33.5 TFfloatField.CheckRange . . . . .	372
10.33.6 TFfloatField.Value . . . . .	372
10.33.7 TFfloatField.Currency . . . . .	372
10.33.8 TFfloatField.MaxValue . . . . .	373
10.33.9 TFfloatField.MinValue . . . . .	373
10.33.10 TFfloatField.Precision . . . . .	373
10.34 TFMTBCDField . . . . .	374
10.34.1 Description . . . . .	374
10.34.2 Method overview . . . . .	374
10.34.3 Property overview . . . . .	374
10.34.4 TFMTBCDField.Create . . . . .	374
10.34.5 TFMTBCDField.CheckRange . . . . .	374
10.34.6 TFMTBCDField.Value . . . . .	375
10.34.7 TFMTBCDField.Precision . . . . .	375
10.34.8 TFMTBCDField.Currency . . . . .	375
10.34.9 TFMTBCDField.MaxValue . . . . .	375
10.34.10 TFMTBCDField.MinValue . . . . .	376
10.34.11 TFMTBCDField.Size . . . . .	376

10.35 TGraphicField . . . . .	376
10.35.1 Description . . . . .	376
10.35.2 Method overview . . . . .	376
10.35.3 TGraphicField.Create . . . . .	376
10.36 TGuidField . . . . .	377
10.36.1 Description . . . . .	377
10.36.2 Method overview . . . . .	377
10.36.3 Property overview . . . . .	377
10.36.4 TG guidField.Create . . . . .	377
10.36.5 TG guidField.AsGuid . . . . .	377
10.37 TIndexDef . . . . .	378
10.37.1 Description . . . . .	378
10.37.2 Method overview . . . . .	378
10.37.3 Property overview . . . . .	378
10.37.4 TIndexDef.Create . . . . .	378
10.37.5 TIndexDef.Expression . . . . .	378
10.37.6 TIndexDef.Fields . . . . .	379
10.37.7 TIndexDef.CaseInsFields . . . . .	379
10.37.8 TIndexDef.DescFields . . . . .	379
10.37.9 TIndexDef.Options . . . . .	380
10.37.10 TIndexDef.Source . . . . .	380
10.38 TIndexDefs . . . . .	380
10.38.1 Description . . . . .	380
10.38.2 Method overview . . . . .	380
10.38.3 Property overview . . . . .	380
10.38.4 TIndexDefs.Create . . . . .	381
10.38.5 TIndexDefs.Add . . . . .	381
10.38.6 TIndexDefs.AddIndexDef . . . . .	381
10.38.7 TIndexDefs.Find . . . . .	381
10.38.8 TIndexDefs.FindIndexForFields . . . . .	382
10.38.9 TIndexDefs.GetIndexForFields . . . . .	382
10.38.10 TIndexDefs.Update . . . . .	382
10.38.11 TIndexDefs.Items . . . . .	382
10.39 TIntegerField . . . . .	383
10.39.1 Description . . . . .	383
10.40 TLargeintField . . . . .	383
10.40.1 Description . . . . .	383
10.40.2 Method overview . . . . .	383
10.40.3 Property overview . . . . .	383
10.40.4 TL largeintField.Create . . . . .	383

10.40.5 TLargeintField.CheckRange . . . . .	383
10.40.6 TLargeintField.Value . . . . .	384
10.40.7 TLargeintField.MaxValue . . . . .	384
10.40.8 TLargeintField.MinValue . . . . .	384
10.41 TLongintField . . . . .	385
10.41.1 Description . . . . .	385
10.41.2 Method overview . . . . .	385
10.41.3 Property overview . . . . .	385
10.41.4 TLongintField.Create . . . . .	385
10.41.5 TLongintField.CheckRange . . . . .	385
10.41.6 TLongintField.Value . . . . .	386
10.41.7 TLongintField.MaxValue . . . . .	386
10.41.8 TLongintField.MinValue . . . . .	386
10.42 TLookupList . . . . .	386
10.42.1 Description . . . . .	386
10.42.2 Method overview . . . . .	387
10.42.3 TLookupList.Create . . . . .	387
10.42.4 TLookupList.Destroy . . . . .	387
10.42.5 TLookupList.Add . . . . .	387
10.42.6 TLookupList.Clear . . . . .	387
10.42.7 TLookupList.FirstKeyByValue . . . . .	388
10.42.8 TLookupList.ValueOfKey . . . . .	388
10.42.9 TLookupList.ValuesToStrings . . . . .	388
10.43 TMasterDataLink . . . . .	388
10.43.1 Description . . . . .	388
10.43.2 Method overview . . . . .	389
10.43.3 Property overview . . . . .	389
10.43.4 TMasterDataLink.Create . . . . .	389
10.43.5 TMasterDataLink.Destroy . . . . .	389
10.43.6 TMasterDataLink.FieldNames . . . . .	389
10.43.7 TMasterDataLink.Fields . . . . .	390
10.43.8 TMasterDataLink.OnMasterChange . . . . .	390
10.43.9 TMasterDataLink.OnMasterDisable . . . . .	390
10.44 TMasterParamsDataLink . . . . .	390
10.44.1 Description . . . . .	390
10.44.2 Method overview . . . . .	391
10.44.3 Property overview . . . . .	391
10.44.4 TMasterParamsDataLink.Create . . . . .	391
10.44.5 TMasterParamsDataLink.RefreshParamNames . . . . .	391
10.44.6 TMasterParamsDataLink.CopyParamsFromMaster . . . . .	391

10.44.7 TMasterParamsDataLink.Params	392
10.45 TMemoField	392
10.45.1 Description	392
10.45.2 Method overview	392
10.45.3 Property overview	392
10.45.4 TMemoField.Create	392
10.45.5 TMemoField.Transliterate	393
10.46 TNamedItem	393
10.46.1 Description	393
10.46.2 Property overview	393
10.46.3 TNamedItem.DisplayName	393
10.46.4 TNamedItem.Name	393
10.47 TNumericField	394
10.47.1 Description	394
10.47.2 Method overview	394
10.47.3 Property overview	394
10.47.4 TNumericField.Create	394
10.47.5 TNumericField.Alignment	394
10.47.6 TNumericField.DisplayFormat	395
10.47.7 TNumericField.EditFormat	395
10.48 TParam	395
10.48.1 Description	395
10.48.2 Method overview	396
10.48.3 Property overview	396
10.48.4 TParam.Create	396
10.48.5 TParam.Assign	397
10.48.6 TParam.AssignField	397
10.48.7 TParam.AssignToField	397
10.48.8 TParam.AssignFieldValue	398
10.48.9 TParam.AssignFromField	398
10.48.10 TParam.Clear	398
10.48.11 TParam.GetData	398
10.48.12 TParam.GetDataSize	399
10.48.13 TParam.LoadFromFile	399
10.48.14 TParam.LoadFromStream	399
10.48.15 TParam.SetBlobData	399
10.48.16 TParam.SetData	400
10.48.17 TParam.AsBlob	400
10.48.18 TParam.AsBoolean	400
10.48.19 TParam.AsCurrency	400

10.48.20TParam.AsDate . . . . .	401
10.48.21TPParam.AsDateTime . . . . .	401
10.48.22TPParam.AsFloat . . . . .	401
10.48.23TPParam.AsInteger . . . . .	401
10.48.24TPParam.AsLargeInt . . . . .	402
10.48.25TPParam.AsMemo . . . . .	402
10.48.26TPParam.AsSmallInt . . . . .	402
10.48.27TPParam.AsString . . . . .	402
10.48.28TPParam.AsTime . . . . .	403
10.48.29TPParam.AsWord . . . . .	403
10.48.30TPParam.AsFMTBCD . . . . .	403
10.48.31TPParam.Bound . . . . .	403
10.48.32TPParam.Dataset . . . . .	404
10.48.33TPParam.IsNull . . . . .	404
10.48.34TPParam.NativeStr . . . . .	404
10.48.35TPParam.Text . . . . .	404
10.48.36TPParam.Value . . . . .	405
10.48.37TPParam.AsWideString . . . . .	405
10.48.38TPParam.DataType . . . . .	405
10.48.39TPParam.Name . . . . .	405
10.48.40TPParam.NumericScale . . . . .	406
10.48.41TPParam.ParamType . . . . .	406
10.48.42TPParam.Precision . . . . .	406
10.48.43TPParam.Size . . . . .	407
10.49TPParams . . . . .	407
10.49.1 Description . . . . .	407
10.49.2 Method overview . . . . .	407
10.49.3 Property overview . . . . .	408
10.49.4 TParams.Create . . . . .	408
10.49.5 TParams.AddParam . . . . .	408
10.49.6 TParams.AssignValues . . . . .	408
10.49.7 TParams.CreateParam . . . . .	408
10.49.8 TParams.FindParam . . . . .	409
10.49.9 TParams.GetParamList . . . . .	409
10.49.10TParams.AreEqual . . . . .	409
10.49.11TPParams.ParamByName . . . . .	410
10.49.12TPParams.ParseSQL . . . . .	410
10.49.13TPParams.RemoveParam . . . . .	411
10.49.14TPParams.CopyParamValuesFromDataset . . . . .	411
10.49.15TPParams.Dataset . . . . .	411

10.49.16 TParams.Items . . . . .	412
10.49.17 TParams.ParamValues . . . . .	412
10.50 TSmallintField . . . . .	412
10.50.1 Description . . . . .	412
10.50.2 Method overview . . . . .	412
10.50.3 TSmallintField.Create . . . . .	413
10.51 TStringField . . . . .	413
10.51.1 Description . . . . .	413
10.51.2 Method overview . . . . .	413
10.51.3 Property overview . . . . .	413
10.51.4 TStringField.Create . . . . .	413
10.51.5 TStringField.SetFieldType . . . . .	414
10.51.6 TStringField.FixedChar . . . . .	414
10.51.7 TStringField.Transliterate . . . . .	414
10.51.8 TStringField.Value . . . . .	414
10.51.9 TStringField.EditMask . . . . .	415
10.51.10 TStringField.Size . . . . .	415
10.52 TTimeField . . . . .	415
10.52.1 Description . . . . .	415
10.52.2 Method overview . . . . .	415
10.52.3 TTimeField.Create . . . . .	416
10.53 TVarBytesField . . . . .	416
10.53.1 Description . . . . .	416
10.53.2 Method overview . . . . .	416
10.53.3 TVarBytesField.Create . . . . .	416
10.54 TVariantField . . . . .	416
10.54.1 Description . . . . .	416
10.54.2 Method overview . . . . .	417
10.54.3 TVariantField.Create . . . . .	417
10.55 TWideMemoField . . . . .	417
10.55.1 Description . . . . .	417
10.55.2 Method overview . . . . .	417
10.55.3 Property overview . . . . .	417
10.55.4 TWideMemoField.Create . . . . .	417
10.55.5 TWideMemoField.Value . . . . .	418
10.56 TWideStringField . . . . .	418
10.56.1 Description . . . . .	418
10.56.2 Method overview . . . . .	418
10.56.3 Property overview . . . . .	418
10.56.4 TWideStringField.Create . . . . .	418

10.56.5 TWideStringField.SetFieldType . . . . .	419
10.56.6 TWideStringField.Value . . . . .	419
10.57 TWordField . . . . .	419
10.57.1 Description . . . . .	419
10.57.2 Method overview . . . . .	419
10.57.3 TWordField.Create . . . . .	419
<b>11 Reference for unit 'dbugint'</b> . . . . .	<b>420</b>
11.1 Overview . . . . .	420
11.2 Writing a debug server . . . . .	420
11.3 Constants, types and variables . . . . .	420
11.3.1 Resource strings . . . . .	420
11.3.2 Constants . . . . .	421
11.3.3 Types . . . . .	421
11.4 Procedures and functions . . . . .	421
11.4.1 GetDebuggingEnabled . . . . .	421
11.4.2 InitDebugClient . . . . .	422
11.4.3 SendBoolean . . . . .	422
11.4.4 SendDateTime . . . . .	422
11.4.5 SendDebug . . . . .	422
11.4.6 SendDebugEx . . . . .	423
11.4.7 SendDebugFmt . . . . .	423
11.4.8 SendDebugFmtEx . . . . .	423
11.4.9 SendInteger . . . . .	424
11.4.10 SendMethodEnter . . . . .	424
11.4.11 SendMethodExit . . . . .	424
11.4.12 SendPointer . . . . .	425
11.4.13 SendSeparator . . . . .	425
11.4.14 SetDebuggingEnabled . . . . .	425
11.4.15 StartDebugServer . . . . .	425
<b>12 Reference for unit 'dbugmsg'</b> . . . . .	<b>427</b>
12.1 Used units . . . . .	427
12.2 Overview . . . . .	427
12.3 Constants, types and variables . . . . .	427
12.3.1 Constants . . . . .	427
12.3.2 Types . . . . .	428
12.4 Procedures and functions . . . . .	428
12.4.1 DebugMessageName . . . . .	428
12.4.2 ReadDebugMessageFromStream . . . . .	428
12.4.3 WriteDebugMessageToStream . . . . .	429

<b>13 Reference for unit 'eventlog'</b>	<b>430</b>
13.1 Used units . . . . .	430
13.2 Overview . . . . .	430
13.3 Constants, types and variables . . . . .	430
13.3.1 Resource strings . . . . .	430
13.3.2 Types . . . . .	431
13.4 ELogError . . . . .	431
13.4.1 Description . . . . .	431
13.5 TEventLog . . . . .	432
13.5.1 Description . . . . .	432
13.5.2 Method overview . . . . .	432
13.5.3 Property overview . . . . .	432
13.5.4 TEventLog.Destroy . . . . .	432
13.5.5 TEventLog.EventTypeToString . . . . .	433
13.5.6 TEventLog.RegisterMessageFile . . . . .	433
13.5.7 TEventLog.UnRegisterMessageFile . . . . .	434
13.5.8 TEventLog.Pause . . . . .	434
13.5.9 TEventLog.Resume . . . . .	434
13.5.10 TEventLog.Log . . . . .	434
13.5.11 TEventLog.Warning . . . . .	435
13.5.12 TEventLog.Error . . . . .	435
13.5.13 TEventLog.Debug . . . . .	435
13.5.14 TEventLog.Info . . . . .	436
13.5.15 TEventLog.AppendContent . . . . .	436
13.5.16 TEventLog.Identification . . . . .	436
13.5.17 TEventLog.LogType . . . . .	436
13.5.18 TEventLog.Active . . . . .	437
13.5.19 TEventLog.RaiseExceptionOnError . . . . .	437
13.5.20 TEventLog.DefaultEventType . . . . .	437
13.5.21 TEventLog.FileName . . . . .	437
13.5.22 TEventLog.TimeStampFormat . . . . .	438
13.5.23 TEventLog.CustomLogType . . . . .	438
13.5.24 TEventLog.EventIDOffset . . . . .	438
13.5.25 TEventLog.OnGetCustomCategory . . . . .	439
13.5.26 TEventLog.OnGetCustomEventID . . . . .	439
13.5.27 TEventLog.OnGetCustomEvent . . . . .	439
13.5.28 TEventLog.Paused . . . . .	440
<b>14 Reference for unit 'ezcg'</b>	<b>441</b>
14.1 Used units . . . . .	441

14.2 Overview . . . . .	441
14.3 Constants, types and variables . . . . .	441
14.3.1 Constants . . . . .	441
14.4 ECGIException . . . . .	441
14.4.1 Description . . . . .	441
14.5 TEZcgi . . . . .	442
14.5.1 Description . . . . .	442
14.5.2 Method overview . . . . .	442
14.5.3 Property overview . . . . .	442
14.5.4 TEZcgi.Create . . . . .	442
14.5.5 TEZcgi.Destroy . . . . .	442
14.5.6 TEZcgi.Run . . . . .	443
14.5.7 TEZcgi.WriteContent . . . . .	443
14.5.8 TEZcgi.PutLine . . . . .	443
14.5.9 TEZcgi.GetValue . . . . .	444
14.5.10 TEZcgi.DoPost . . . . .	444
14.5.11 TEZcgi.DoGet . . . . .	444
14.5.12 TEZcgi.Values . . . . .	444
14.5.13 TEZcgi.Names . . . . .	445
14.5.14 TEZcgi.Variables . . . . .	445
14.5.15 TEZcgi.VariableCount . . . . .	446
14.5.16 TEZcgi.Name . . . . .	446
14.5.17 TEZcgi.Email . . . . .	446
<b>15 Reference for unit 'fpjson'</b> . . . . .	<b>447</b>
15.1 Used units . . . . .	447
15.2 Overview . . . . .	447
15.3 Constants, types and variables . . . . .	449
15.3.1 Constants . . . . .	449
15.3.2 Types . . . . .	449
15.4 Procedures and functions . . . . .	452
15.4.1 CreateJSON . . . . .	452
15.4.2 CreateJSONArray . . . . .	453
15.4.3 CreateJSONObject . . . . .	453
15.4.4 GetJSON . . . . .	453
15.4.5 GetJSONInstanceType . . . . .	454
15.4.6 GetJSONParserHandler . . . . .	454
15.4.7 JSONStringToString . . . . .	454
15.4.8 JSONTypeName . . . . .	454
15.4.9 SetJSONInstanceType . . . . .	455

15.4.10 SetJSONParserHandler . . . . .	455
15.4.11 StringToJSONString . . . . .	455
15.5 EJSON . . . . .	456
15.5.1 Description . . . . .	456
15.6 TBaseJSONEnumerator . . . . .	456
15.6.1 Description . . . . .	456
15.6.2 Method overview . . . . .	456
15.6.3 Property overview . . . . .	456
15.6.4 TBaseJSONEnumerator.GetCurrent . . . . .	456
15.6.5 TBaseJSONEnumerator.MoveNext . . . . .	456
15.6.6 TBaseJSONEnumerator.Current . . . . .	457
15.7 TJSONArray . . . . .	457
15.7.1 Description . . . . .	457
15.7.2 Method overview . . . . .	457
15.7.3 Property overview . . . . .	457
15.7.4 TJSONArray.Create . . . . .	458
15.7.5 TJSONArray.Destroy . . . . .	458
15.7.6 TJSONArray.JSONType . . . . .	458
15.7.7 TJSONArray.Clone . . . . .	458
15.7.8 TJSONArray.Iterate . . . . .	459
15.7.9 TJSONArray.IndexOf . . . . .	459
15.7.10 TJSONArray.GetEnumerator . . . . .	459
15.7.11 TJSONArray.Clear . . . . .	459
15.7.12 TJSONArray.Add . . . . .	460
15.7.13 TJSONArray.Delete . . . . .	460
15.7.14 TJSONArray.Exchange . . . . .	460
15.7.15 TJSONArray.Extract . . . . .	461
15.7.16 TJSONArray.Insert . . . . .	461
15.7.17 TJSONArray.Move . . . . .	461
15.7.18 TJSONArray.Remove . . . . .	462
15.7.19 TJSONArray.Items . . . . .	462
15.7.20 TJSONArray.Types . . . . .	462
15.7.21 TJSONArray.Nulls . . . . .	462
15.7.22 TJSONArray.Integers . . . . .	463
15.7.23 TJSONArray.Int64s . . . . .	463
15.7.24 TJSONArray.Strings . . . . .	464
15.7.25 TJSONArray.Floats . . . . .	464
15.7.26 TJSONArrayBOOLEANS . . . . .	464
15.7.27 TJSONArray.Arrays . . . . .	465
15.7.28 TJSONArray.Objects . . . . .	465

15.8 TJSONBoolean . . . . .	465
15.8.1 Description . . . . .	465
15.8.2 Method overview . . . . .	466
15.8.3 TJSONBoolean.Create . . . . .	466
15.8.4 TJSONBoolean.JSONType . . . . .	466
15.8.5 TJSONBoolean.Clear . . . . .	466
15.8.6 TJSONBoolean.Clone . . . . .	466
15.9 TJSONData . . . . .	467
15.9.1 Description . . . . .	467
15.9.2 Method overview . . . . .	467
15.9.3 Property overview . . . . .	467
15.9.4 TJSONData.Create . . . . .	467
15.9.5 TJSONData.JSONType . . . . .	468
15.9.6 TJSONData.Clear . . . . .	468
15.9.7 TJSONData.GetEnumerator . . . . .	468
15.9.8 TJSONData.FindPath . . . . .	469
15.9.9 TJSONData.GetPath . . . . .	471
15.9.10 TJSONData.Clone . . . . .	471
15.9.11 TJSONData.FormatJSON . . . . .	471
15.9.12 TJSONData.Count . . . . .	472
15.9.13 TJSONData.Items . . . . .	472
15.9.14 TJSONData.Value . . . . .	472
15.9.15 TJSONData.AsString . . . . .	472
15.9.16 TJSONData.AsFloat . . . . .	473
15.9.17 TJSONData.AsInteger . . . . .	473
15.9.18 TJSONData.AsInt64 . . . . .	474
15.9.19 TJSONData.AsBoolean . . . . .	474
15.9.20 TJSONData.IsNull . . . . .	474
15.9.21 TJSONData.AsJSON . . . . .	475
15.10 TJSONFloatNumber . . . . .	475
15.10.1 Description . . . . .	475
15.10.2 Method overview . . . . .	475
15.10.3 TJSONFloatNumber.Create . . . . .	475
15.10.4 TJSONFloatNumber.NumberType . . . . .	475
15.10.5 TJSONFloatNumber.Clear . . . . .	476
15.10.6 TJSONFloatNumber.Clone . . . . .	476
15.11 TJSONInt64Number . . . . .	476
15.11.1 Description . . . . .	476
15.11.2 Method overview . . . . .	476
15.11.3 TJSONInt64Number.Create . . . . .	476

15.11.4 TJSONInt64Number.NumberType . . . . .	477
15.11.5 TJSONInt64Number.Clear . . . . .	477
15.11.6 TJSONInt64Number.Clone . . . . .	477
15.12 TJSONIntegerNumber . . . . .	477
15.12.1 Description . . . . .	477
15.12.2 Method overview . . . . .	477
15.12.3 TJSONIntegerNumber.Create . . . . .	478
15.12.4 TJSONIntegerNumber.NumberType . . . . .	478
15.12.5 TJSONIntegerNumber.Clear . . . . .	478
15.12.6 TJSONIntegerNumber.Clone . . . . .	478
15.13 TJSONNull . . . . .	478
15.13.1 Description . . . . .	478
15.13.2 Method overview . . . . .	479
15.13.3 TJSONNull.JSONType . . . . .	479
15.13.4 TJSONNull.Clear . . . . .	479
15.13.5 TJSONNull.Clone . . . . .	479
15.14 TJSONNumber . . . . .	479
15.14.1 Description . . . . .	479
15.14.2 Method overview . . . . .	480
15.14.3 TJSONNumber.JSONType . . . . .	480
15.14.4 TJSONNumber.NumberType . . . . .	480
15.15 TJSONObject . . . . .	480
15.15.1 Description . . . . .	480
15.15.2 Method overview . . . . .	481
15.15.3 Property overview . . . . .	481
15.15.4 TJSONObject.Create . . . . .	481
15.15.5 TJSONObject.Destroy . . . . .	482
15.15.6 TJSONObject.JSONType . . . . .	482
15.15.7 TJSONObject.Clone . . . . .	482
15.15.8 TJSONObject.GetEnumerator . . . . .	483
15.15.9 TJSONObject.Iterate . . . . .	483
15.15.10 TJSONObject.IndexOf . . . . .	483
15.15.11 TJSONObject.IndexOfName . . . . .	483
15.15.12 TJSONObject.Find . . . . .	484
15.15.13 TJSONObject.Get . . . . .	484
15.15.14 TJSONObject.Clear . . . . .	484
15.15.15 TJSONObject.Add . . . . .	485
15.15.16 TJSONObject.Delete . . . . .	485
15.15.17 TJSONObject.Remove . . . . .	485
15.15.18 TJSONObject.Extract . . . . .	486

15.15.19 TJSONObject.Names . . . . .	486
15.15.20 TJSONObject.Elements . . . . .	486
15.15.21 TJSONObject.Types . . . . .	487
15.15.22 TJSONObject.Nulls . . . . .	487
15.15.23 TJSONObject.Floats . . . . .	487
15.15.24 TJSONObject.Integers . . . . .	487
15.15.25 TJSONObject.Int64s . . . . .	488
15.15.26 TJSONObject.Strings . . . . .	488
15.15.27 TJSONObjectBOOLEANS . . . . .	488
15.15.28 TJSONObject.Arrays . . . . .	489
15.15.29 TJSONObject.Objects . . . . .	489
15.16 TJSONString . . . . .	489
15.16.1 Description . . . . .	489
15.16.2 Method overview . . . . .	489
15.16.3 TJSONString.Create . . . . .	490
15.16.4 TJSONString.JSONType . . . . .	490
15.16.5 TJSONString.Clear . . . . .	490
15.16.6 TJSONString.Clone . . . . .	490
<b>16 Reference for unit 'fpTimer'</b> . . . . .	<b>491</b>
16.1 Used units . . . . .	491
16.2 Overview . . . . .	491
16.3 Constants, types and variables . . . . .	491
16.3.1 Types . . . . .	491
16.3.2 Variables . . . . .	491
16.4 TFPCustomTimer . . . . .	492
16.4.1 Description . . . . .	492
16.4.2 Method overview . . . . .	492
16.4.3 TFPCustomTimer.Create . . . . .	492
16.4.4 TFPCustomTimer.Destroy . . . . .	492
16.4.5 TFPCustomTimer.StartTimer . . . . .	493
16.4.6 TFPCustomTimer.StopTimer . . . . .	493
16.5 TFPTimer . . . . .	493
16.5.1 Description . . . . .	493
16.5.2 Property overview . . . . .	493
16.5.3 TFPTimer.Enabled . . . . .	493
16.5.4 TFPTimer.Interval . . . . .	494
16.5.5 TFPTimer.OnTimer . . . . .	494
16.6 TFPTimerDriver . . . . .	494
16.6.1 Description . . . . .	494

16.6.2	Method overview . . . . .	494
16.6.3	Property overview . . . . .	494
16.6.4	TFPTimerDriver.Create . . . . .	495
16.6.5	TFPTimerDriver.StartTimer . . . . .	495
16.6.6	TFPTimerDriver.StopTimer . . . . .	495
16.6.7	TFPTimerDriver.Timer . . . . .	495
<b>17</b>	<b>Reference for unit 'gettext'</b>	<b>496</b>
17.1	Used units . . . . .	496
17.2	Overview . . . . .	496
17.3	Constants, types and variables . . . . .	496
17.3.1	Constants . . . . .	496
17.3.2	Types . . . . .	496
17.4	Procedures and functions . . . . .	497
17.4.1	GetLanguageIDs . . . . .	497
17.4.2	TranslateResourceStrings . . . . .	498
17.4.3	TranslateUnitResourceStrings . . . . .	498
17.5	EMOFLError . . . . .	498
17.5.1	Description . . . . .	498
17.6	TMOFile . . . . .	498
17.6.1	Description . . . . .	498
17.6.2	Method overview . . . . .	499
17.6.3	TMOFile.Create . . . . .	499
17.6.4	TMOFile.Destroy . . . . .	499
17.6.5	TMOFile.Translate . . . . .	499
<b>18</b>	<b>Reference for unit 'IBConnection'</b>	<b>500</b>
18.1	Used units . . . . .	500
18.2	Constants, types and variables . . . . .	500
18.2.1	Constants . . . . .	500
18.2.2	Types . . . . .	500
18.3	EIBDatabaseError . . . . .	501
18.3.1	Description . . . . .	501
18.4	TIBConnection . . . . .	501
18.4.1	Description . . . . .	501
18.4.2	Method overview . . . . .	502
18.4.3	Property overview . . . . .	502
18.4.4	TIBConnection.Create . . . . .	502
18.4.5	TIBConnection.GetConnectionInfo . . . . .	502
18.4.6	TIBConnection.CreateDB . . . . .	502
18.4.7	TIBConnection.DropDB . . . . .	503

18.4.8 TIBConnection.BlobSegmentSize . . . . .	503
18.4.9 TIBConnection.ODSMajorVersion . . . . .	504
18.4.10 TIBConnection.DatabaseName . . . . .	504
18.4.11 TIBConnection.Dialect . . . . .	504
18.4.12 TIBConnection.KeepConnection . . . . .	504
18.4.13 TIBConnection.LoginPrompt . . . . .	505
18.4.14 TIBConnection.Params . . . . .	505
18.4.15 TIBConnection.OnLogin . . . . .	505
18.5 TIBConnectionDef . . . . .	506
18.5.1 Description . . . . .	506
18.5.2 Method overview . . . . .	506
18.5.3 TIBConnectionDef.TypeName . . . . .	506
18.5.4 TIBConnectionDef.ConnectionClass . . . . .	506
18.5.5 TIBConnectionDef.Description . . . . .	506
18.5.6 TIBConnectionDef.DefaultLibraryName . . . . .	507
18.5.7 TIBConnectionDef.LoadFunction . . . . .	507
18.5.8 TIBConnectionDef.UnLoadFunction . . . . .	507
18.5.9 TIBConnectionDef.LoadedLibraryName . . . . .	507
18.6 TIBCursor . . . . .	507
18.6.1 Description . . . . .	507
18.7 TIBTrans . . . . .	507
18.7.1 Description . . . . .	507
<b>19 Reference for unit 'idea'</b> . . . . .	<b>508</b>
19.1 Used units . . . . .	508
19.2 Overview . . . . .	508
19.3 Constants, types and variables . . . . .	508
19.3.1 Constants . . . . .	508
19.3.2 Types . . . . .	509
19.4 Procedures and functions . . . . .	509
19.4.1 CipherIdea . . . . .	509
19.4.2 DeKeyIdea . . . . .	509
19.4.3 EnKeyIdea . . . . .	510
19.5 EIDEAError . . . . .	510
19.5.1 Description . . . . .	510
19.6 TIDEADeCryptStream . . . . .	510
19.6.1 Description . . . . .	510
19.6.2 Method overview . . . . .	510
19.6.3 TIDEADeCryptStream.Create . . . . .	511
19.6.4 TIDEADeCryptStream.Read . . . . .	511

19.6.5	TIDEADeCryptStream.Seek . . . . .	511
19.7	TIDEAEncryptStream . . . . .	512
19.7.1	Description . . . . .	512
19.7.2	Method overview . . . . .	512
19.7.3	TIDEAEncryptStream.Create . . . . .	512
19.7.4	TIDEAEncryptStream.Destroy . . . . .	512
19.7.5	TIDEAEncryptStream.Write . . . . .	513
19.7.6	TIDEAEncryptStream.Seek . . . . .	513
19.7.7	TIDEAEncryptStream.Flush . . . . .	513
19.8	TIDEAStream . . . . .	513
19.8.1	Description . . . . .	513
19.8.2	Method overview . . . . .	514
19.8.3	Property overview . . . . .	514
19.8.4	TIDEAStream.Create . . . . .	514
19.8.5	TIDEAStream.Key . . . . .	514
<b>20</b>	<b>Reference for unit 'inicoll'</b>	<b>515</b>
20.1	Used units . . . . .	515
20.2	Overview . . . . .	515
20.3	Constants, types and variables . . . . .	515
20.3.1	Constants . . . . .	515
20.4	EIniCol . . . . .	516
20.4.1	Description . . . . .	516
20.5	TIniCollection . . . . .	516
20.5.1	Description . . . . .	516
20.5.2	Method overview . . . . .	516
20.5.3	Property overview . . . . .	516
20.5.4	TIniCollection.Load . . . . .	516
20.5.5	TIniCollection.Save . . . . .	517
20.5.6	TIniCollection.SaveToIni . . . . .	517
20.5.7	TIniCollection.SaveToFile . . . . .	517
20.5.8	TIniCollection.LoadFromIni . . . . .	518
20.5.9	TIniCollection.LoadFromFile . . . . .	518
20.5.10	TIniCollection.Prefix . . . . .	518
20.5.11	TIniCollection.SectionPrefix . . . . .	519
20.5.12	TIniCollection.FileName . . . . .	519
20.5.13	TIniCollection.GlobalSection . . . . .	519
20.6	TIniCollectionItem . . . . .	520
20.6.1	Description . . . . .	520
20.6.2	Method overview . . . . .	520

20.6.3	Property overview	520
20.6.4	TIniCollectionItem.SaveToIni	520
20.6.5	TIniCollectionItem.LoadFromIni	520
20.6.6	TIniCollectionItem.SaveToFile	521
20.6.7	TIniCollectionItem.LoadFromFile	521
20.6.8	TIniCollectionItem.SectionName	521
20.7	TNamedIniCollection	522
20.7.1	Description	522
20.7.2	Method overview	522
20.7.3	Property overview	522
20.7.4	TNamedIniCollection.IndexOfUserData	522
20.7.5	TNamedIniCollection.IndexOfName	522
20.7.6	TNamedIniCollection.FindByName	523
20.7.7	TNamedIniCollection.FindByUserData	523
20.7.8	TNamedIniCollection.NamedItems	523
20.8	TNamedIniCollectionItem	523
20.8.1	Description	523
20.8.2	Property overview	523
20.8.3	TNamedIniCollectionItem.UserData	524
20.8.4	TNamedIniCollectionItem.Name	524
<b>21</b>	<b>Reference for unit 'IniFiles'</b>	<b>525</b>
21.1	Used units	525
21.2	Overview	525
21.3	TCustomIniFile	525
21.3.1	Description	525
21.3.2	Method overview	526
21.3.3	Property overview	526
21.3.4	TCustomIniFile.Create	526
21.3.5	TCustomIniFile.Destroy	527
21.3.6	TCustomIniFile.SectionExists	527
21.3.7	TCustomIniFile.ReadString	527
21.3.8	TCustomIniFile.WriteString	528
21.3.9	TCustomIniFile.ReadInteger	528
21.3.10	TCustomIniFile.WriteInteger	528
21.3.11	TCustomIniFile.ReadInt64	528
21.3.12	TCustomIniFile.WriteInt64	529
21.3.13	TCustomIniFile.ReadBool	529
21.3.14	TCustomIniFile.WriteBool	529
21.3.15	TCustomIniFile.ReadDate	530

21.3.16 TCustomIniFile.ReadDateTime . . . . .	530
21.3.17 TCustomIniFile.ReadFloat . . . . .	530
21.3.18 TCustomIniFile.ReadTime . . . . .	530
21.3.19 TCustomIniFile.ReadBinaryStream . . . . .	531
21.3.20 TCustomIniFile.WriteDate . . . . .	531
21.3.21 TCustomIniFile.WriteDateTime . . . . .	531
21.3.22 TCustomIniFile.WriteFloat . . . . .	532
21.3.23 TCustomIniFile.WriteTime . . . . .	532
21.3.24 TCustomIniFile.WriteBinaryStream . . . . .	532
21.3.25 TCustomIniFile.ReadSection . . . . .	532
21.3.26 TCustomIniFile.ReadSections . . . . .	533
21.3.27 TCustomIniFile.ReadSectionValues . . . . .	533
21.3.28 TCustomIniFile.EraseSection . . . . .	533
21.3.29 TCustomIniFile.DeleteKey . . . . .	534
21.3.30 TCustomIniFile.UpdateFile . . . . .	534
21.3.31 TCustomIniFile.ValueExists . . . . .	534
21.3.32 TCustomIniFile.FileName . . . . .	534
21.3.33 TCustomIniFile.EscapeLineFeeds . . . . .	535
21.3.34 TCustomIniFile.CaseSensitive . . . . .	535
21.3.35 TCustomIniFile.StripQuotes . . . . .	535
21.4 THashedStringList . . . . .	535
21.4.1 Description . . . . .	535
21.4.2 Method overview . . . . .	536
21.4.3 THashedStringList.Destroy . . . . .	536
21.4.4 THashedStringList.IndexOf . . . . .	536
21.4.5 THashedStringList.IndexOfName . . . . .	536
21.5 TIniFile . . . . .	536
21.5.1 Description . . . . .	536
21.5.2 Method overview . . . . .	537
21.5.3 Property overview . . . . .	537
21.5.4 TIniFile.Create . . . . .	537
21.5.5 TIniFile.Destroy . . . . .	537
21.5.6 TIniFile.ReadString . . . . .	538
21.5.7 TIniFile.WriteString . . . . .	538
21.5.8 TIniFile.ReadSection . . . . .	538
21.5.9 TIniFile.ReadSectionRaw . . . . .	538
21.5.10 TIniFile.ReadSections . . . . .	539
21.5.11 TIniFile.ReadSectionValues . . . . .	539
21.5.12 TIniFile.EraseSection . . . . .	539
21.5.13 TIniFile.DeleteKey . . . . .	539

21.5.14 TIniFile.UpdateFile . . . . .	540
21.5.15 TIniFile.Stream . . . . .	540
21.5.16 TIniFile.CacheUpdates . . . . .	540
21.6 TIniFileKey . . . . .	540
21.6.1 Description . . . . .	540
21.6.2 Method overview . . . . .	541
21.6.3 Property overview . . . . .	541
21.6.4 TIniFileKey.Create . . . . .	541
21.6.5 TIniFileKey.Ident . . . . .	541
21.6.6 TIniFileKey.Value . . . . .	541
21.7 TIniFileKeyList . . . . .	542
21.7.1 Description . . . . .	542
21.7.2 Method overview . . . . .	542
21.7.3 Property overview . . . . .	542
21.7.4 TIniFileKeyList.Destroy . . . . .	542
21.7.5 TIniFileKeyList.Clear . . . . .	542
21.7.6 TIniFileKeyList.Items . . . . .	542
21.8 TIniFileSection . . . . .	543
21.8.1 Description . . . . .	543
21.8.2 Method overview . . . . .	543
21.8.3 Property overview . . . . .	543
21.8.4 TIniFileSection.Empty . . . . .	543
21.8.5 TIniFileSection.Create . . . . .	543
21.8.6 TIniFileSection.Destroy . . . . .	543
21.8.7 TIniFileSection.Name . . . . .	544
21.8.8 TIniFileSection.KeyList . . . . .	544
21.9 TIniFileSectionList . . . . .	544
21.9.1 Description . . . . .	544
21.9.2 Method overview . . . . .	544
21.9.3 Property overview . . . . .	544
21.9.4 TIniFileSectionList.Destroy . . . . .	545
21.9.5 TIniFileSectionList.Clear . . . . .	545
21.9.6 TIniFileSectionList.Items . . . . .	545
21.10 TMemIniFile . . . . .	545
21.10.1 Description . . . . .	545
21.10.2 Method overview . . . . .	545
21.10.3 TMemIniFile.Create . . . . .	546
21.10.4 TMemIniFile.Clear . . . . .	546
21.10.5 TMemIniFile.GetStrings . . . . .	546
21.10.6 TMemIniFile.Rename . . . . .	546

21.10.7 TMemIniFile.SetStrings . . . . .	547
<b>22 Reference for unit 'iostream'</b>	<b>548</b>
22.1 Used units . . . . .	548
22.2 Overview . . . . .	548
22.3 Constants, types and variables . . . . .	548
22.3.1 Types . . . . .	548
22.4 EIOStreamError . . . . .	549
22.4.1 Description . . . . .	549
22.5 TIOStream . . . . .	549
22.5.1 Description . . . . .	549
22.5.2 Method overview . . . . .	549
22.5.3 TIOStream.Create . . . . .	549
22.5.4 TIOStream.Read . . . . .	549
22.5.5 TIOStream.Write . . . . .	550
22.5.6 TIOStream.Seek . . . . .	550
<b>23 Reference for unit 'libtar'</b>	<b>551</b>
23.1 Used units . . . . .	551
23.2 Overview . . . . .	551
23.3 Constants, types and variables . . . . .	551
23.3.1 Constants . . . . .	551
23.3.2 Types . . . . .	552
23.4 Procedures and functions . . . . .	554
23.4.1 ClearDirRec . . . . .	554
23.4.2 ConvertFilename . . . . .	554
23.4.3 FileTimeGMT . . . . .	554
23.4.4 PermissionString . . . . .	554
23.5 TTarArchive . . . . .	555
23.5.1 Description . . . . .	555
23.5.2 Method overview . . . . .	555
23.5.3 TTarArchive.Create . . . . .	555
23.5.4 TTarArchive.Destroy . . . . .	555
23.5.5 TTarArchive.Reset . . . . .	555
23.5.6 TTarArchive.FindNext . . . . .	556
23.5.7 TTarArchive.ReadFile . . . . .	556
23.5.8 TTarArchive.GetFilePos . . . . .	556
23.5.9 TTarArchive.SetFilePos . . . . .	557
23.6 TTarWriter . . . . .	557
23.6.1 Description . . . . .	557
23.6.2 Method overview . . . . .	557

23.6.3	Property overview . . . . .	557
23.6.4	TTarWriter.Create . . . . .	557
23.6.5	TTarWriter.Destroy . . . . .	558
23.6.6	TTarWriter.AddFile . . . . .	558
23.6.7	TTarWriter.AddStream . . . . .	558
23.6.8	TTarWriter.AddString . . . . .	559
23.6.9	TTarWriter.AddDir . . . . .	559
23.6.10	TTarWriter.AddSymbolicLink . . . . .	559
23.6.11	TTarWriter.AddLink . . . . .	560
23.6.12	TTarWriter.AddVolumeHeader . . . . .	560
23.6.13	TTarWriter.Finalize . . . . .	560
23.6.14	TTarWriter.Permissions . . . . .	560
23.6.15	TTarWriter.UID . . . . .	561
23.6.16	TTarWriter.GID . . . . .	561
23.6.17	TTarWriter.UserName . . . . .	561
23.6.18	TTarWriter.GroupName . . . . .	561
23.6.19	TTarWriter.Mode . . . . .	562
23.6.20	TTarWriter.Magic . . . . .	562
<b>24</b>	<b>Reference for unit 'mssqlconn'</b>	<b>563</b>
24.1	Used units . . . . .	563
24.2	Overview . . . . .	563
24.3	Constants, types and variables . . . . .	563
24.3.1	Types . . . . .	563
24.3.2	Variables . . . . .	564
24.4	EMSSQLDatabaseError . . . . .	564
24.4.1	Description . . . . .	564
24.5	TMSSQLConnection . . . . .	564
24.5.1	Description . . . . .	564
24.5.2	Method overview . . . . .	565
24.5.3	Property overview . . . . .	565
24.5.4	TMSSQLConnection.Create . . . . .	565
24.5.5	TMSSQLConnection.GetConnectionString . . . . .	565
24.5.6	TMSSQLConnection.CreateDB . . . . .	565
24.5.7	TMSSQLConnection.DropDB . . . . .	565
24.5.8	TMSSQLConnection.Password . . . . .	565
24.5.9	TMSSQLConnection.Transaction . . . . .	566
24.5.10	TMSSQLConnection.UserName . . . . .	566
24.5.11	TMSSQLConnection.CharSet . . . . .	566
24.5.12	TMSSQLConnection.HostName . . . . .	566

24.5.13 TMSSQLConnection.Connected . . . . .	567
24.5.14 TMSSQLConnection.Role . . . . .	567
24.5.15 TMSSQLConnection.DatabaseName . . . . .	567
24.5.16 TMSSQLConnection.KeepConnection . . . . .	567
24.5.17 TMSSQLConnection.LoginPrompt . . . . .	567
24.5.18 TMSSQLConnection.Params . . . . .	567
24.5.19 TMSSQLConnection.OnLogin . . . . .	568
24.6 TMSSQLConnectionDef . . . . .	568
24.6.1 Method overview . . . . .	568
24.6.2 TMSSQLConnectionDef.TypeName . . . . .	568
24.6.3 TMSSQLConnectionDef.ConnectionClass . . . . .	568
24.6.4 TMSSQLConnectionDef.Description . . . . .	568
24.6.5 TMSSQLConnectionDef.DefaultLibraryName . . . . .	568
24.6.6 TMSSQLConnectionDef.LoadFunction . . . . .	568
24.6.7 TMSSQLConnectionDef.UnLoadFunction . . . . .	569
24.6.8 TMSSQLConnectionDef.LoadedLibraryName . . . . .	569
24.7 TSybaseConnection . . . . .	569
24.7.1 Description . . . . .	569
24.7.2 Method overview . . . . .	569
24.7.3 TSybaseConnection.Create . . . . .	569
24.8 TSybaseConnectionDef . . . . .	569
24.8.1 Method overview . . . . .	569
24.8.2 TSybaseConnectionDef.TypeName . . . . .	570
24.8.3 TSybaseConnectionDef.ConnectionClass . . . . .	570
24.8.4 TSybaseConnectionDef.Description . . . . .	570
<b>25 Reference for unit 'Pipes'</b>	<b>571</b>
25.1 Used units . . . . .	571
25.2 Overview . . . . .	571
25.3 Constants, types and variables . . . . .	571
25.3.1 Constants . . . . .	571
25.4 Procedures and functions . . . . .	571
25.4.1 CreatePipeHandles . . . . .	571
25.4.2 CreatePipeStreams . . . . .	572
25.5 EPipeCreation . . . . .	572
25.5.1 Description . . . . .	572
25.6 EPipeError . . . . .	572
25.6.1 Description . . . . .	572
25.7 EPipeSeek . . . . .	572
25.7.1 Description . . . . .	572

25.8 TInputPipeStream . . . . .	573
25.8.1 Description . . . . .	573
25.8.2 Method overview . . . . .	573
25.8.3 Property overview . . . . .	573
25.8.4 TInputPipeStream.Destroy . . . . .	573
25.8.5 TInputPipeStream.Write . . . . .	573
25.8.6 TInputPipeStream.Seek . . . . .	574
25.8.7 TInputPipeStream.Read . . . . .	574
25.8.8 TInputPipeStream.NumBytesAvailable . . . . .	574
25.9 TOutputPipeStream . . . . .	575
25.9.1 Description . . . . .	575
25.9.2 Method overview . . . . .	575
25.9.3 TOutputPipeStream.Destroy . . . . .	575
25.9.4 TOutputPipeStream.Seek . . . . .	575
25.9.5 TOutputPipeStream.Read . . . . .	575
<b>26 Reference for unit 'pooledmm'</b>	<b>576</b>
26.1 Used units . . . . .	576
26.2 Overview . . . . .	576
26.3 Constants, types and variables . . . . .	576
26.3.1 Types . . . . .	576
26.4 TNonFreePooledMemManager . . . . .	577
26.4.1 Description . . . . .	577
26.4.2 Method overview . . . . .	577
26.4.3 Property overview . . . . .	577
26.4.4 TNonFreePooledMemManager.Clear . . . . .	577
26.4.5 TNonFreePooledMemManager.Create . . . . .	577
26.4.6 TNonFreePooledMemManager.Destroy . . . . .	578
26.4.7 TNonFreePooledMemManager.NewItem . . . . .	578
26.4.8 TNonFreePooledMemManager.EnumerateItems . . . . .	578
26.4.9 TNonFreePooledMemManager.ItemSize . . . . .	578
26.5 TPooledMemManager . . . . .	579
26.5.1 Description . . . . .	579
26.5.2 Method overview . . . . .	579
26.5.3 Property overview . . . . .	579
26.5.4 TPooledMemManager.Clear . . . . .	579
26.5.5 TPooledMemManager.Create . . . . .	579
26.5.6 TPooledMemManager.Destroy . . . . .	580
26.5.7 TPooledMemManager.MinimumFreeCount . . . . .	580
26.5.8 TPooledMemManager.MaximumFreeCountRatio . . . . .	580

26.5.9 TPooledMemManager.Count . . . . .	580
26.5.10 TPooledMemManager.FreeCount . . . . .	581
26.5.11 TPooledMemManager.AllocatedCount . . . . .	581
26.5.12 TPooledMemManager.FreedCount . . . . .	581
<b>27 Reference for unit 'process'</b>	<b>582</b>
27.1 Used units . . . . .	582
27.2 Overview . . . . .	582
27.3 Constants, types and variables . . . . .	582
27.3.1 Types . . . . .	582
27.3.2 Variables . . . . .	584
27.4 Procedures and functions . . . . .	585
27.4.1 CommandToList . . . . .	585
27.4.2 DetectXTerm . . . . .	585
27.4.3 RunCommand . . . . .	585
27.4.4 RunCommandIndir . . . . .	586
27.5 EProcess . . . . .	586
27.5.1 Description . . . . .	586
27.6 TProcess . . . . .	586
27.6.1 Description . . . . .	586
27.6.2 Method overview . . . . .	587
27.6.3 Property overview . . . . .	588
27.6.4 TProcess.Create . . . . .	588
27.6.5 TProcess.Destroy . . . . .	589
27.6.6 TProcess.Execute . . . . .	589
27.6.7 TProcess.CloseInput . . . . .	589
27.6.8 TProcess.CloseOutput . . . . .	590
27.6.9 TProcess.CloseStderr . . . . .	590
27.6.10 TProcess.Resume . . . . .	590
27.6.11 TProcess.Suspend . . . . .	590
27.6.12 TProcess.Terminate . . . . .	591
27.6.13 TProcess.WaitOnExit . . . . .	591
27.6.14 TProcess.WindowRect . . . . .	591
27.6.15 TProcess.Handle . . . . .	591
27.6.16 TProcess.ProcessHandle . . . . .	592
27.6.17 TProcess.ThreadHandle . . . . .	592
27.6.18 TProcess.ProcessID . . . . .	592
27.6.19 TProcess.ThreadID . . . . .	593
27.6.20 TProcess.Input . . . . .	593
27.6.21 TProcess.Output . . . . .	593

27.6.22 TProcess.Stderr . . . . .	594
27.6.23 TProcess.ExitStatus . . . . .	594
27.6.24 TProcess.InheritHandles . . . . .	594
27.6.25 TProcess.OnForkEvent . . . . .	595
27.6.26 TProcess.PipeBufferSize . . . . .	595
27.6.27 TProcess.Active . . . . .	595
27.6.28 TProcess.ApplicationName . . . . .	595
27.6.29 TProcess.CommandLine . . . . .	596
27.6.30 TProcess.Executable . . . . .	596
27.6.31 TProcess.Parameters . . . . .	597
27.6.32 TProcess.ConsoleTitle . . . . .	597
27.6.33 TProcess.CurrentDirectory . . . . .	598
27.6.34 TProcess.Desktop . . . . .	598
27.6.35 TProcess.Environment . . . . .	598
27.6.36 TProcess.Options . . . . .	598
27.6.37 TProcess.Priority . . . . .	599
27.6.38 TProcess.StartupOptions . . . . .	600
27.6.39 TProcess.Running . . . . .	600
27.6.40 TProcess.ShowWindow . . . . .	601
27.6.41 TProcess.WindowColumns . . . . .	601
27.6.42 TProcess.WindowHeight . . . . .	601
27.6.43 TProcess.WindowLeft . . . . .	602
27.6.44 TProcess.WindowRows . . . . .	602
27.6.45 TProcess.WindowTop . . . . .	602
27.6.46 TProcess.WindowWidth . . . . .	603
27.6.47 TProcess.FillAttribute . . . . .	603
27.6.48 TProcess.XTermProgram . . . . .	603
<b>28 Reference for unit 'rttiutils'</b> . . . . .	<b>604</b>
28.1 Used units . . . . .	604
28.2 Overview . . . . .	604
28.3 Constants, types and variables . . . . .	604
28.3.1 Constants . . . . .	604
28.3.2 Types . . . . .	604
28.3.3 Variables . . . . .	605
28.4 Procedures and functions . . . . .	605
28.4.1 CreateStoredItem . . . . .	605
28.4.2 ParseStoredItem . . . . .	606
28.4.3 UpdateStoredList . . . . .	606
28.5 TPropInfoList . . . . .	606

28.5.1	Description . . . . .	606
28.5.2	Method overview . . . . .	607
28.5.3	Property overview . . . . .	607
28.5.4	TPropInfoList.Create . . . . .	607
28.5.5	TPropInfoList.Destroy . . . . .	607
28.5.6	TPropInfoList.Contains . . . . .	607
28.5.7	TPropInfoList.Find . . . . .	608
28.5.8	TPropInfoList.Delete . . . . .	608
28.5.9	TPropInfoList.Intersect . . . . .	608
28.5.10	TPropInfoList.Count . . . . .	608
28.5.11	TPropInfoList.Items . . . . .	609
28.6	TPropsStorage . . . . .	609
28.6.1	Description . . . . .	609
28.6.2	Method overview . . . . .	609
28.6.3	Property overview . . . . .	609
28.6.4	TPropsStorage.StoreAnyProperty . . . . .	609
28.6.5	TPropsStorage.LoadAnyProperty . . . . .	610
28.6.6	TPropsStorage.StoreProperties . . . . .	610
28.6.7	TPropsStorage.LoadProperties . . . . .	610
28.6.8	TPropsStorage.LoadObjectsProps . . . . .	611
28.6.9	TPropsStorage.StoreObjectsProps . . . . .	611
28.6.10	TPropsStorage.AObject . . . . .	612
28.6.11	TPropsStorage.Prefix . . . . .	612
28.6.12	TPropsStorage.Section . . . . .	612
28.6.13	TPropsStorage.OnReadString . . . . .	613
28.6.14	TPropsStorage.OnWriteString . . . . .	613
28.6.15	TPropsStorage.OnEraseSection . . . . .	613
<b>29</b>	<b>Reference for unit 'simpleipc'</b>	<b>614</b>
29.1	Used units . . . . .	614
29.2	Overview . . . . .	614
29.3	Constants, types and variables . . . . .	614
29.3.1	Resource strings . . . . .	614
29.3.2	Constants . . . . .	615
29.3.3	Types . . . . .	615
29.3.4	Variables . . . . .	616
29.4	EIPCError . . . . .	616
29.4.1	Description . . . . .	616
29.5	TIPCCClientComm . . . . .	616
29.5.1	Description . . . . .	616

29.5.2 Method overview . . . . .	616
29.5.3 Property overview . . . . .	616
29.5.4 TIPCCClientComm.Create . . . . .	617
29.5.5 TIPCCClientComm.Connect . . . . .	617
29.5.6 TIPCCClientComm.Disconnect . . . . .	617
29.5.7 TIPCCClientComm.ServerRunning . . . . .	618
29.5.8 TIPCCClientComm.SendMessage . . . . .	618
29.5.9 TIPCCClientComm.Owner . . . . .	618
29.6 TIPCServerComm . . . . .	618
29.6.1 Description . . . . .	618
29.6.2 Method overview . . . . .	619
29.6.3 Property overview . . . . .	619
29.6.4 TIPCServerComm.Create . . . . .	619
29.6.5 TIPCServerComm.StartServer . . . . .	619
29.6.6 TIPCServerComm.StopServer . . . . .	619
29.6.7 TIPCServerComm.PeekMessage . . . . .	620
29.6.8 TIPCServerComm.ReadMessage . . . . .	620
29.6.9 TIPCServerComm.Owner . . . . .	620
29.6.10 TIPCServerComm.InstanceID . . . . .	621
29.7 TSimpleIPC . . . . .	621
29.7.1 Description . . . . .	621
29.7.2 Property overview . . . . .	621
29.7.3 TSsimpleIPC.Active . . . . .	621
29.7.4 TSsimpleIPC.ServerID . . . . .	621
29.8 TSsimpleIPCCClient . . . . .	622
29.8.1 Description . . . . .	622
29.8.2 Method overview . . . . .	622
29.8.3 Property overview . . . . .	622
29.8.4 TSsimpleIPCCClient.Create . . . . .	622
29.8.5 TSsimpleIPCCClient.Destroy . . . . .	622
29.8.6 TSsimpleIPCCClient.Connect . . . . .	623
29.8.7 TSsimpleIPCCClient.Disconnect . . . . .	623
29.8.8 TSsimpleIPCCClient.ServerRunning . . . . .	623
29.8.9 TSsimpleIPCCClient.SendMessage . . . . .	624
29.8.10 TSsimpleIPCCClient.SendStringMessage . . . . .	624
29.8.11 TSsimpleIPCCClient.SendStringMessageFmt . . . . .	624
29.8.12 TSsimpleIPCCClient.ServerInstance . . . . .	624
29.9 TSsimpleIPCServer . . . . .	625
29.9.1 Description . . . . .	625
29.9.2 Method overview . . . . .	625

29.9.3	Property overview	625
29.9.4	TSimpleIPCServer.Create	625
29.9.5	TSimpleIPCServer.Destroy	626
29.9.6	TSimpleIPCServer.StartServer	626
29.9.7	TSimpleIPCServer.StopServer	626
29.9.8	TSimpleIPCServer.PeekMessage	626
29.9.9	TSimpleIPCServer.GetMessageData	627
29.9.10	TSimpleIPCServer.StringMessage	627
29.9.11	TSimpleIPCServer.MsgType	627
29.9.12	TSimpleIPCServer.MsgData	628
29.9.13	TSimpleIPCServer.InstanceID	628
29.9.14	TSimpleIPCServer.Global	628
29.9.15	TSimpleIPCServer.OnMessage	628
<b>30</b>	<b>Reference for unit 'sqldb'</b>	<b>630</b>
30.1	Used units	630
30.2	Overview	630
30.3	Using SQLDB to access databases	631
30.4	Using the universal TSQLConnector type	633
30.5	Retrieving Schema Information	634
30.6	Automatic generation of update SQL statements	634
30.7	Using parameters	635
30.8	Constants, types and variables	636
30.8.1	Constants	636
30.8.2	Types	637
30.8.3	Variables	640
30.9	Procedures and functions	640
30.9.1	GetConnectionDef	640
30.9.2	GetConnectionList	640
30.9.3	RegisterConnection	641
30.9.4	UnRegisterConnection	641
30.10	TConnectionDef	641
30.10.1	Description	641
30.10.2	Method overview	642
30.10.3	TConnectionDef.TypeName	642
30.10.4	TConnectionDef.ConnectionClass	642
30.10.5	TConnectionDef.Description	642
30.10.6	TConnectionDef.DefaultLibraryName	643
30.10.7	TConnectionDef.LoadFunction	643
30.10.8	TConnectionDef.UnLoadFunction	643

30.10.9 TConnectionDef.LoadedLibraryName . . . . .	643
30.10.10 TConnectionDef.ApplyParams . . . . .	644
30.11 TCustomSQLQuery . . . . .	644
30.11.1 Description . . . . .	644
30.11.2 Method overview . . . . .	644
30.11.3 Property overview . . . . .	644
30.11.4 TCustomSQLQuery.Prepare . . . . .	644
30.11.5 TCustomSQLQuery.UnPrepare . . . . .	645
30.11.6 TCustomSQLQuery.ExecSQL . . . . .	645
30.11.7 TCustomSQLQuery.Create . . . . .	646
30.11.8 TCustomSQLQuery.Destroy . . . . .	646
30.11.9 TCustomSQLQuery.SetSchemaInfo . . . . .	646
30.11.10 TCustomSQLQuery.RowsAffected . . . . .	647
30.11.11 ITCustomSQLQuery.ParamByName . . . . .	647
30.11.12 TCustomSQLQuery.Prepared . . . . .	647
30.12 TCustomSQLStatement . . . . .	648
30.12.1 Description . . . . .	648
30.12.2 Method overview . . . . .	648
30.12.3 Property overview . . . . .	648
30.12.4 TCustomSQLStatement.Create . . . . .	648
30.12.5 TCustomSQLStatement.Destroy . . . . .	649
30.12.6 TCustomSQLStatement.Prepare . . . . .	649
30.12.7 TCustomSQLStatement.Execute . . . . .	649
30.12.8 TCustomSQLStatement.Unprepare . . . . .	649
30.12.9 TCustomSQLStatement.ParamByName . . . . .	650
30.12.10 TCustomSQLStatement.RowsAffected . . . . .	650
30.12.11 ITCustomSQLStatement.Prepared . . . . .	650
30.13 TServerIndexDefs . . . . .	651
30.13.1 Description . . . . .	651
30.13.2 Method overview . . . . .	651
30.13.3 TServerIndexDefs.Create . . . . .	651
30.13.4 TServerIndexDefs.Update . . . . .	651
30.14 TSQlConnection . . . . .	651
30.14.1 Description . . . . .	651
30.14.2 Method overview . . . . .	652
30.14.3 Property overview . . . . .	652
30.14.4 TSQlConnection.Create . . . . .	652
30.14.5 TSQlConnection.Destroy . . . . .	653
30.14.6 TSQlConnection.StartTransaction . . . . .	653
30.14.7 TSQlConnection.EndTransaction . . . . .	653

30.14.8 TSQLConnection.ExecuteDirect . . . . .	653
30.14.9 TSQLConnection.GetTableNames . . . . .	654
30.14.10 TSQLConnection.GetProcedureNames . . . . .	654
30.14.11 TSQLConnection.GetFieldNames . . . . .	654
30.14.12 TSQLConnection.GetSchemaNames . . . . .	655
30.14.13 TSQLConnection.GetConnectionInfo . . . . .	655
30.14.14 TSQLConnection.CreateDB . . . . .	655
30.14.15 TSQLConnection.DropDB . . . . .	655
30.14.16 TSQLConnection.Handle . . . . .	656
30.14.17 TSQLConnection.FieldNameQuoteChars . . . . .	656
30.14.18 TSQLConnection.ConnOptions . . . . .	656
30.14.19 TSQLConnection.Password . . . . .	657
30.14.20 TSQLConnection.Transaction . . . . .	657
30.14.21 TSQLConnection.UserName . . . . .	657
30.14.22 TSQLConnection.CharSet . . . . .	658
30.14.23 TSQLConnection.HostName . . . . .	658
30.14.24 TSQLConnection.OnLog . . . . .	658
30.14.25 TSQLConnection.LogEvents . . . . .	659
30.14.26 TSQLConnection.Connected . . . . .	659
30.14.27 TSQLConnection.Role . . . . .	659
30.14.28 TSQLConnection.DatabaseName . . . . .	660
30.14.29 TSQLConnection.KeepConnection . . . . .	660
30.14.30 TSQLConnection.LoginPrompt . . . . .	660
30.14.31 TSQLConnection.Params . . . . .	660
30.14.32 TSQLConnection.OnLogin . . . . .	661
30.15 TSQLConnector . . . . .	661
30.15.1 Description . . . . .	661
30.15.2 Property overview . . . . .	661
30.15.3 TSQLConnector.ConnectorType . . . . .	661
30.16 TSQLCursor . . . . .	662
30.16.1 Description . . . . .	662
30.17 TSQLHandle . . . . .	662
30.17.1 Description . . . . .	662
30.18 TSQLQuery . . . . .	662
30.18.1 Description . . . . .	662
30.18.2 Property overview . . . . .	663
30.18.3 TSQLQuery.SchemaType . . . . .	664
30.18.4 TSQLQuery.StatementType . . . . .	664
30.18.5 TSQLQuery.MaxIndexesCount . . . . .	664
30.18.6 TSQLQuery.FieldDefs . . . . .	664

30.18.7 TSQLQuery.Active . . . . .	665
30.18.8 TSQLQuery.AutoCalcFields . . . . .	665
30.18.9 TSQLQuery.Filter . . . . .	665
30.18.10 TSQLQuery.Filtered . . . . .	665
30.18.11 TSQLQuery.AfterCancel . . . . .	665
30.18.12 TSQLQuery.AfterClose . . . . .	665
30.18.13 TSQLQuery.AfterDelete . . . . .	665
30.18.14 TSQLQuery.AfterEdit . . . . .	666
30.18.15 TSQLQuery.AfterInsert . . . . .	666
30.18.16 TSQLQuery.AfterOpen . . . . .	666
30.18.17 TSQLQuery.AfterPost . . . . .	666
30.18.18 TSQLQuery.AfterScroll . . . . .	666
30.18.19 TSQLQuery.BeforeCancel . . . . .	666
30.18.20 TSQLQuery.BeforeClose . . . . .	666
30.18.21 TSQLQuery.BeforeDelete . . . . .	667
30.18.22 TSQLQuery.BeforeEdit . . . . .	667
30.18.23 TSQLQuery.BeforeInsert . . . . .	667
30.18.24 TSQLQuery.BeforeOpen . . . . .	667
30.18.25 TSQLQuery.BeforePost . . . . .	667
30.18.26 TSQLQuery.BeforeScroll . . . . .	667
30.18.27 TSQLQuery.OnCalcFields . . . . .	667
30.18.28 TSQLQuery.OnDeleteError . . . . .	668
30.18.29 TSQLQuery.OnEditError . . . . .	668
30.18.30 TSQLQuery.OnFilterRecord . . . . .	668
30.18.31 TSQLQuery.OnNewRecord . . . . .	668
30.18.32 TSQLQuery.OnPostError . . . . .	668
30.18.33 TSQLQuery.Database . . . . .	668
30.18.34 TSQLQuery.Transaction . . . . .	669
30.18.35 TSQLQuery.ReadOnly . . . . .	669
30.18.36 TSQLQuery.SQL . . . . .	669
30.18.37 TSQLQuery.UpdateSQL . . . . .	669
30.18.38 TSQLQuery.InsertSQL . . . . .	670
30.18.39 TSQLQuery.DeleteSQL . . . . .	670
30.18.40 TSQLQuery.IndexDefs . . . . .	671
30.18.41 TSQLQuery.Params . . . . .	671
30.18.42 TSQLQuery.ParamCheck . . . . .	671
30.18.43 TSQLQuery.ParseSQL . . . . .	672
30.18.44 TSQLQuery.UpdateMode . . . . .	672
30.18.45 TSQLQuery.UsePrimaryKeyAsKey . . . . .	672
30.18.46 TSQLQuery.DataSource . . . . .	673

30.18.4 <code>TSQLQuery.ServerFilter</code>	673
30.18.4 <code>TSQLQuery.ServerFiltered</code>	673
30.18.4 <code>TSQLQuery.ServerIndexDefs</code>	674
30.19 <code>TSQLScript</code>	674
30.19.1 <code>Description</code>	674
30.19.2 <code>Method overview</code>	674
30.19.3 <code>Property overview</code>	674
30.19.4 <code>TSQLScript.Create</code>	674
30.19.5 <code>TSQLScript.Destroy</code>	675
30.19.6 <code>TSQLScript.Execute</code>	675
30.19.7 <code>TSQLScript.ExecuteScript</code>	675
30.19.8 <code>TSQLScript.DataBase</code>	676
30.19.9 <code>TSQLScript.Transaction</code>	676
30.19.10 <code>TSQLScript.OnDirective</code>	676
30.19.11 <code>TSQLScript.Directives</code>	676
30.19.12 <code>TSQLScriptDefines</code>	677
30.19.13 <code>TSQLScriptScript</code>	677
30.19.14 <code>TSQLScriptTerminator</code>	677
30.19.15 <code>TSQLScriptCommentsinSQL</code>	678
30.19.16 <code>TSQLScriptUseSetTerm</code>	678
30.19.17 <code>TSQLScriptUseCommit</code>	678
30.19.18 <code>TSQLScriptUseDefines</code>	679
30.19.19 <code>TSQLScriptOnException</code>	679
30.20 <code>TSQLStatement</code>	680
30.20.1 <code>Description</code>	680
30.20.2 <code>Property overview</code>	680
30.20.3 <code>TSQLStatement.Database</code>	680
30.20.4 <code>TSQLStatement.DataSource</code>	680
30.20.5 <code>TSQLStatement.ParamCheck</code>	681
30.20.6 <code>TSQLStatement.Params</code>	681
30.20.7 <code>TSQLStatement.ParseSQL</code>	681
30.20.8 <code>TSQLStatement.SQL</code>	681
30.20.9 <code>TSQLStatement.Transaction</code>	682
30.21 <code>TSQLTransaction</code>	682
30.21.1 <code>Description</code>	682
30.21.2 <code>Method overview</code>	682
30.21.3 <code>Property overview</code>	683
30.21.4 <code>TSQLTransaction.Commit</code>	683
30.21.5 <code>TSQLTransaction.CommitRetaining</code>	683
30.21.6 <code>TSQLTransaction.Rollback</code>	683

30.21.7 TSQLTransaction.RollbackRetaining . . . . .	684
30.21.8 TSQLTransaction.StartTransaction . . . . .	684
30.21.9 TSQLTransaction.Create . . . . .	685
30.21.10 TSQLTransaction.Destroy . . . . .	685
30.21.11 ITSQLTransaction.EndTransaction . . . . .	685
30.21.12 ITSQLTransaction.Handle . . . . .	685
30.21.13 ITSQLTransaction.Action . . . . .	685
30.21.14 ITSQLTransaction.Database . . . . .	686
30.21.15 ITSQLTransaction.Params . . . . .	686
<b>31 Reference for unit 'streamcoll'</b> . . . . .	<b>687</b>
31.1 Used units . . . . .	687
31.2 Overview . . . . .	687
31.3 Procedures and functions . . . . .	687
31.3.1 ColReadBoolean . . . . .	687
31.3.2 ColReadCurrency . . . . .	688
31.3.3 ColReadDateTime . . . . .	688
31.3.4 ColReadFloat . . . . .	688
31.3.5 ColReadInteger . . . . .	688
31.3.6 ColReadString . . . . .	689
31.3.7 ColWriteBoolean . . . . .	689
31.3.8 ColWriteCurrency . . . . .	689
31.3.9 ColWriteDateTime . . . . .	689
31.3.10 ColWriteFloat . . . . .	690
31.3.11 ColWriteInteger . . . . .	690
31.3.12 ColWriteString . . . . .	690
31.4 EStreamColl . . . . .	690
31.4.1 Description . . . . .	690
31.5 TStreamCollection . . . . .	690
31.5.1 Description . . . . .	690
31.5.2 Method overview . . . . .	691
31.5.3 Property overview . . . . .	691
31.5.4 TStreamCollection.LoadFromStream . . . . .	691
31.5.5 TStreamCollection.SaveToStream . . . . .	691
31.5.6 TStreamCollection.Streaming . . . . .	691
31.6 TStreamCollectionItem . . . . .	692
31.6.1 Description . . . . .	692
<b>32 Reference for unit 'streamex'</b> . . . . .	<b>693</b>
32.1 Used units . . . . .	693
32.2 Overview . . . . .	693

32.3	TBmdirBinaryObjectReader . . . . .	693
32.3.1	Description . . . . .	693
32.3.2	Property overview . . . . .	693
32.3.3	TBmdirBinaryObjectReader.Position . . . . .	693
32.4	TBmdirBinaryObjectWriter . . . . .	694
32.4.1	Description . . . . .	694
32.4.2	Property overview . . . . .	694
32.4.3	TBmdirBinaryObjectWriter.Position . . . . .	694
32.5	TDelphiReader . . . . .	694
32.5.1	Description . . . . .	694
32.5.2	Method overview . . . . .	694
32.5.3	Property overview . . . . .	695
32.5.4	TDelphiReader.GetDriver . . . . .	695
32.5.5	TDelphiReader.ReadStr . . . . .	695
32.5.6	TDelphiReader.Read . . . . .	695
32.5.7	TDelphiReader.Position . . . . .	695
32.6	TDelphiWriter . . . . .	696
32.6.1	Description . . . . .	696
32.6.2	Method overview . . . . .	696
32.6.3	Property overview . . . . .	696
32.6.4	TDelphiWriter.GetDriver . . . . .	696
32.6.5	TDelphiWriter.FlushBuffer . . . . .	696
32.6.6	TDelphiWriter.Write . . . . .	696
32.6.7	TDelphiWriter.WriteString . . . . .	697
32.6.8	TDelphiWriter.writeValue . . . . .	697
32.6.9	TDelphiWriter.Position . . . . .	697
32.7	TStreamHelper . . . . .	697
32.7.1	Description . . . . .	697
32.7.2	Method overview . . . . .	698
32.7.3	TStreamHelper.ReadWordLE . . . . .	698
32.7.4	TStreamHelper.ReadDWordLE . . . . .	698
32.7.5	TStreamHelper.ReadQWordLE . . . . .	698
32.7.6	TStreamHelper.WriteWordLE . . . . .	699
32.7.7	TStreamHelper.WriteDWordLE . . . . .	699
32.7.8	TStreamHelper.WriteQWordLE . . . . .	699
32.7.9	TStreamHelper.ReadWordBE . . . . .	699
32.7.10	TStreamHelper.ReadDWordBE . . . . .	700
32.7.11	TStreamHelper.ReadQWordBE . . . . .	700
32.7.12	TStreamHelper.WriteWordBE . . . . .	700
32.7.13	TStreamHelper.WriteDWordBE . . . . .	700

32.7.14 TStreamHelper.WriteQWordBE . . . . .	701
<b>33 Reference for unit 'StreamIO'</b>	<b>702</b>
33.1 Used units . . . . .	702
33.2 Overview . . . . .	702
33.3 Procedures and functions . . . . .	702
33.3.1 AssignStream . . . . .	702
33.3.2 GetStream . . . . .	703
<b>34 Reference for unit 'syncobjs'</b>	<b>704</b>
34.1 Used units . . . . .	704
34.2 Overview . . . . .	704
34.3 Constants, types and variables . . . . .	704
34.3.1 Constants . . . . .	704
34.3.2 Types . . . . .	704
34.4 TCriticalSection . . . . .	705
34.4.1 Description . . . . .	705
34.4.2 Method overview . . . . .	705
34.4.3 TCriticalSection.Acquire . . . . .	706
34.4.4 TCriticalSection.Release . . . . .	706
34.4.5 TCriticalSection.Enter . . . . .	706
34.4.6 TCriticalSection.TryEnter . . . . .	706
34.4.7 TCriticalSection.Leave . . . . .	707
34.4.8 TCriticalSection.Create . . . . .	707
34.4.9 TCriticalSection.Destroy . . . . .	707
34.5 TEventObject . . . . .	707
34.5.1 Description . . . . .	707
34.5.2 Method overview . . . . .	708
34.5.3 Property overview . . . . .	708
34.5.4 TEventObject.Create . . . . .	708
34.5.5 TEventObject.destroy . . . . .	708
34.5.6 TEventObject.ResetEvent . . . . .	708
34.5.7 TEventObject.SetEvent . . . . .	709
34.5.8 TEventObject.WaitFor . . . . .	709
34.5.9 TEventObject.ManualReset . . . . .	709
34.6 THandleObject . . . . .	709
34.6.1 Description . . . . .	709
34.6.2 Method overview . . . . .	710
34.6.3 Property overview . . . . .	710
34.6.4 THandleObject.destroy . . . . .	710
34.6.5 THandleObject.Handle . . . . .	710

34.6.6	THandleObject.LastError . . . . .	710
34.7	TSimpleEvent . . . . .	710
34.7.1	Description . . . . .	710
34.7.2	Method overview . . . . .	711
34.7.3	TSimpleEvent.Create . . . . .	711
34.8	TSynchroObject . . . . .	711
34.8.1	Description . . . . .	711
34.8.2	Method overview . . . . .	711
34.8.3	TSynchroObject.Acquire . . . . .	711
34.8.4	TSynchroObject.Release . . . . .	711
<b>35</b>	<b>Reference for unit 'URIParser'</b>	<b>713</b>
35.1	Overview . . . . .	713
35.2	Constants, types and variables . . . . .	713
35.2.1	Types . . . . .	713
35.3	Procedures and functions . . . . .	713
35.3.1	EncodeURI . . . . .	713
35.3.2	FilenameToURI . . . . .	714
35.3.3	IsAbsoluteURI . . . . .	714
35.3.4	ParseURI . . . . .	714
35.3.5	ResolveRelativeURI . . . . .	715
35.3.6	URIToFilename . . . . .	715
<b>36</b>	<b>Reference for unit 'zipper'</b>	<b>716</b>
36.1	Used units . . . . .	716
36.2	Overview . . . . .	716
36.3	Constants, types and variables . . . . .	716
36.3.1	Constants . . . . .	716
36.3.2	Types . . . . .	717
36.4	EZipError . . . . .	719
36.4.1	Description . . . . .	719
36.5	TCompressor . . . . .	719
36.5.1	Description . . . . .	719
36.5.2	Method overview . . . . .	719
36.5.3	Property overview . . . . .	719
36.5.4	TCompressor.Create . . . . .	719
36.5.5	TCompressor.Compress . . . . .	720
36.5.6	TCompressor.ZipID . . . . .	720
36.5.7	TCompressor.ZipVersionReqd . . . . .	720
36.5.8	TCompressor.ZipBitFlag . . . . .	720
36.5.9	TCompressor.BufferSize . . . . .	720

36.5.10 TCompressor.OnPercent . . . . .	720
36.5.11 TCompressor.OnProgress . . . . .	720
36.5.12 TCompressor.Crc32Val . . . . .	721
36.6 TDeCompressor . . . . .	721
36.6.1 Description . . . . .	721
36.6.2 Method overview . . . . .	721
36.6.3 Property overview . . . . .	721
36.6.4 TDeCompressor.Create . . . . .	721
36.6.5 TDeCompressor.DeCompress . . . . .	721
36.6.6 TDeCompressor.ZipID . . . . .	722
36.6.7 TDeCompressor.BufferSize . . . . .	722
36.6.8 TDeCompressor.OnPercent . . . . .	722
36.6.9 TDeCompressor.OnProgress . . . . .	722
36.6.10 TDeCompressor.Crc32Val . . . . .	722
36.7 TDeflater . . . . .	722
36.7.1 Description . . . . .	722
36.7.2 Method overview . . . . .	723
36.7.3 Property overview . . . . .	723
36.7.4 TDeflater.Create . . . . .	723
36.7.5 TDeflater.Compress . . . . .	723
36.7.6 TDeflater.ZipID . . . . .	723
36.7.7 TDeflater.ZipVersionReqd . . . . .	723
36.7.8 TDeflater.ZipBitFlag . . . . .	723
36.7.9 TDeflater.CompressionLevel . . . . .	723
36.8 TFullZipFileEntries . . . . .	724
36.8.1 Property overview . . . . .	724
36.8.2 TFullZipFileEntries.FullEntries . . . . .	724
36.9 TFullZipFileEntry . . . . .	724
36.9.1 Property overview . . . . .	724
36.9.2 TFullZipFileEntry.CompressMethod . . . . .	724
36.9.3 TFullZipFileEntry.CompressedSize . . . . .	724
36.9.4 TFullZipFileEntry.CRC32 . . . . .	724
36.10 TInflator . . . . .	724
36.10.1 Description . . . . .	724
36.10.2 Method overview . . . . .	725
36.10.3 TInflator.Create . . . . .	725
36.10.4 TInflator.DeCompress . . . . .	725
36.10.5 TInflator.ZipID . . . . .	725
36.11 TShrinker . . . . .	725
36.11.1 Description . . . . .	725

36.11.2 Method overview . . . . .	725
36.11.3 TShrinker.Create . . . . .	725
36.11.4 TShrinker.Destroy . . . . .	726
36.11.5 TShrinker.Compress . . . . .	726
36.11.6 TShrinker.ZipID . . . . .	726
36.11.7 TShrinker.ZipVersionReqd . . . . .	726
36.11.8 TShrinker.ZipBitFlag . . . . .	726
36.12 TUnZipper . . . . .	726
36.12.1 Method overview . . . . .	726
36.12.2 Property overview . . . . .	727
36.12.3 TUnZipper.Create . . . . .	727
36.12.4 TUnZipper.Destroy . . . . .	727
36.12.5 TUnZipper.UnZipAllFiles . . . . .	727
36.12.6 TUnZipper.UnZipFiles . . . . .	728
36.12.7 TUnZipper.Clear . . . . .	728
36.12.8 TUnZipper.Examine . . . . .	728
36.12.9 TUnZipper.BufferSize . . . . .	728
36.12.10 TUnZipper.OnOpenInputStream . . . . .	728
36.12.11 TUnZipper.OnCloseInputStream . . . . .	728
36.12.12 TUnZipper.OnCreateStream . . . . .	729
36.12.13 TUnZipper.OnDoneStream . . . . .	729
36.12.14 TUnZipper.OnPercent . . . . .	729
36.12.15 TUnZipper.OnProgress . . . . .	729
36.12.16 TUnZipper.OnStartFile . . . . .	729
36.12.17 TUnZipper.OnEndFile . . . . .	729
36.12.18 TUnZipper.FileName . . . . .	730
36.12.19 TUnZipper.OutputPath . . . . .	730
36.12.20 TUnZipper.FileComment . . . . .	730
36.12.21 TUnZipper.Files . . . . .	730
36.12.22 TUnZipper.Entries . . . . .	730
36.13 TZipFileEntries . . . . .	730
36.13.1 Description . . . . .	730
36.13.2 Method overview . . . . .	731
36.13.3 Property overview . . . . .	731
36.13.4 TZipFileEntries.AddFileEntry . . . . .	731
36.13.5 TZipFileEntries.AddFileEntries . . . . .	731
36.13.6 TZipFileEntries.Entries . . . . .	731
36.14 TZipFileEntry . . . . .	731
36.14.1 Method overview . . . . .	731
36.14.2 Property overview . . . . .	732

36.14.3 TZipFileEntry.Create . . . . .	732
36.14.4 TZipFileEntry.IsDirectory . . . . .	732
36.14.5 TZipFileEntry.IsLink . . . . .	732
36.14.6 TZipFileEntry.Assign . . . . .	732
36.14.7 TZipFileEntry.Stream . . . . .	732
36.14.8 TZipFileEntry.ArchiveFileName . . . . .	732
36.14.9 TZipFileEntry.DiskFileName . . . . .	733
36.14.10 TZipFileEntry.Size . . . . .	733
36.14.11 TZipFileEntry.DateTime . . . . .	733
36.14.12 TZipFileEntry.OS . . . . .	733
36.14.13 TZipFileEntry.Attributes . . . . .	733
36.14.14 TZipFileEntry.CompressionLevel . . . . .	733
36.15 TZipper . . . . .	734
36.15.1 Method overview . . . . .	734
36.15.2 Property overview . . . . .	734
36.15.3 TZipper.Create . . . . .	734
36.15.4 TZipper.Destroy . . . . .	734
36.15.5 TZipper.ZipAllFiles . . . . .	734
36.15.6 TZipper.SaveToFile . . . . .	735
36.15.7 TZipper.SaveAsStream . . . . .	735
36.15.8 TZipper.ZipFiles . . . . .	735
36.15.9 TZipper.Clear . . . . .	735
36.15.10 TZipper.BufferSize . . . . .	735
36.15.11 TZipper.OnPercent . . . . .	735
36.15.12 TZipper.OnProgress . . . . .	736
36.15.13 TZipper.OnStartFile . . . . .	736
36.15.14 TZipper.OnEndFile . . . . .	736
36.15.15 TZipper.FileName . . . . .	736
36.15.16 TZipper.FileComment . . . . .	736
36.15.17 TZipper.Files . . . . .	736
36.15.18 TZipper.InMemSize . . . . .	736
36.15.19 TZipper.Entries . . . . .	737
<b>37 Reference for unit 'zstream'</b> . . . . .	<b>738</b>
37.1 Used units . . . . .	738
37.2 Overview . . . . .	738
37.3 Constants, types and variables . . . . .	738
37.3.1 Types . . . . .	738
37.4 Ecompressionerror . . . . .	739
37.4.1 Description . . . . .	739

37.5 Edecompressionerror . . . . .	739
37.5.1 Description . . . . .	739
37.6 Egzfileerror . . . . .	739
37.6.1 Description . . . . .	739
37.7 Ezliberror . . . . .	739
37.7.1 Description . . . . .	739
37.8 Tcompressionstream . . . . .	739
37.8.1 Description . . . . .	739
37.8.2 Method overview . . . . .	740
37.8.3 Property overview . . . . .	740
37.8.4 Tcompressionstream.create . . . . .	740
37.8.5 Tcompressionstream.destroy . . . . .	740
37.8.6 Tcompressionstream.write . . . . .	740
37.8.7 Tcompressionstream.flush . . . . .	741
37.8.8 Tcompressionstream.get_compressionrate . . . . .	741
37.8.9 Tcompressionstream.OnProgress . . . . .	741
37.9 Tcustomzlibstream . . . . .	741
37.9.1 Description . . . . .	741
37.9.2 Method overview . . . . .	742
37.9.3 Tcustomzlibstream.create . . . . .	742
37.9.4 Tcustomzlibstream.destroy . . . . .	742
37.10 Tdecompressionstream . . . . .	742
37.10.1 Description . . . . .	742
37.10.2 Method overview . . . . .	742
37.10.3 Property overview . . . . .	743
37.10.4 Tdecompressionstream.create . . . . .	743
37.10.5 Tdecompressionstream.destroy . . . . .	743
37.10.6 Tdecompressionstream.read . . . . .	743
37.10.7 Tdecompressionstream.seek . . . . .	744
37.10.8 Tdecompressionstream.get_compressionrate . . . . .	744
37.10.9 Tdecompressionstream.OnProgress . . . . .	744
37.11 TGZFileStream . . . . .	745
37.11.1 Description . . . . .	745
37.11.2 Method overview . . . . .	745
37.11.3 TGZFileStream.create . . . . .	745
37.11.4 TGZFileStream.read . . . . .	745
37.11.5 TGZFileStream.write . . . . .	746
37.11.6 TGZFileStream.seek . . . . .	746
37.11.7 TGZFileStream.destroy . . . . .	746

## 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 have their own subsections, and for each function or procedure we have the following topics:

**Declaration** The exact declaration of the function.

**Description** What does the procedure exactly do ?

**Errors** What errors can occur.

**See Also** Cross references to other related functions/commands.

## 0.1 Overview

The 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 descendants 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'

Name	Page
Classes	??
System	??
sysutils	??

### 1.2 Overview

The `ascii85` provides an ASCII 85 or base 85 decoding algorithm. It is class and stream based: the `TASCII85DecoderStream` (72) 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` (508) or `base64` (93) 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 TASCIIS85State

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

TASCIIS85State is for internal use, it contains the current state of the decoder.

## 1.4 TASCIIS85DecoderStream

### 1.4.1 Description

TASCIIS85DecoderStream 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

Page	Property	Description
73	Close	Close decoder
73	ClosedP	Check if the state is correct
72	Create	Create new ASCII 85 decoder stream
73	Decode	Decode source byte
73	Destroy	Clean up instance
74	Read	Read data from stream
74	Seek	Set stream position

### 1.4.3 Property overview

Page	Property	Access	Description
74	BExpectBoundary	rw	Expect ĉcharacter

### 1.4.4 TASCIIS85DecoderStream.Create

Synopsis: Create new ASCII 85 decoder stream

Declaration: constructor Create (aStream: TStream)

Visibility: published

Description: Create instantiates a new TASCIIS85DecoderStream instance, and sets aStream as the source stream.

See also: TASCIIS85DecoderStream.Destroy (73)

### 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 ([73](#))

### 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 ([73](#)), TASCII85DecoderStream.Read ([74](#)), TASCII85DecoderStream.Destroy ([73](#))

### 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 ([73](#)), TASCII85DecoderStream.BExpectBoundary ([74](#))

### 1.4.8 TASCII85DecoderStream.Destroy

**Synopsis:** Clean up instance

**Declaration:** destructor Destroy; Override

**Visibility:** public

**Description:** Destroy closes the input stream using Close ([73](#)) and cleans up the TASCII85DecoderStream instance from memory.

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

**See also:** TASCII85DecoderStream.Close ([73](#))

### 1.4.9 TASCI85DecoderStream.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 TASCI85DecoderStream.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: TASCI85DecoderStream.Read ([74](#))

### 1.4.11 TASCI85DecoderStream.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 ([73](#))

## 1.5 TASCI85EncoderStream

### 1.5.1 Description

TASCI85EncoderStream is the counterpart to the TASCI85DecoderStream ([72](#)) decoder stream: what TASCI85EncoderStream encodes, can be decoded by TASCI85DecoderStream ([72](#)).

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: TASCI85EncoderStream.create ([75](#)), TASCI85DecoderStream ([72](#))

### 1.5.2 Method overview

Page	Property	Description
<a href="#">75</a>	Create	Create a new instance of TASCII85EncoderStream
<a href="#">75</a>	Destroy	Flushed the data to the output stream and cleans up the encoder instance.
<a href="#">75</a>	Write	Write data encoded to the destination stream

### 1.5.3 Property overview

Page	Property	Access	Description
<a href="#">76</a>	Boundary	r	Is a boundary delineator written before and after the data
<a href="#">76</a>	Width	r	Width of the lines written to the data stream

### 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 ([74](#)), Width ([76](#)), Boundary ([76](#))

### 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 ([75](#)), TASCII85EncoderStream.Boundary ([76](#))

### 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 ([75](#))

### 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 ([76](#)), Create ([75](#))

### 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 ([76](#)), Create ([75](#))

## 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 ([72](#)) decoder stream will create an instance of this class automatically.

See also: TASCII85DecoderStream ([72](#))

### 1.6.2 Method overview

Page	Property	Description
<a href="#">77</a>	Read	Read data from the internal buffer
<a href="#">77</a>	Write	Write data to the internal buffer

### 1.6.3 Property overview

Page	Property	Access	Description
<a href="#">77</a>	FillCount	r	Number of bytes in buffer
<a href="#">77</a>	Size	r	Size of buffer

### 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 ([77](#)), TASCII85RingBuffer.Read ([77](#)), TASCII85RingBuffer.Size ([77](#))

### 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 ([77](#)), TASCII85RingBuffer.Write ([77](#)), TASCII85RingBuffer.Size ([77](#))

### 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 ([77](#)), TASCII85RingBuffer.Read ([77](#)), TASCII85RingBuffer.Size ([77](#))

### 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 ([77](#))

# Chapter 2

## Reference for unit 'AVL\_Tree'

### 2.1 Used units

Table 2.1: Used units by unit 'AVL\_Tree'

Name	Page
Classes	??
System	??
sysutils	??

### 2.2 Overview

The `avl_tree` unit implements a general-purpose AVL (balanced) tree class: the `TAVLTree` (78) class and it's associated data node class `TAVLTreeNode` (87).

### 2.3 TAVLTree

#### 2.3.1 Description

`TAVLTree` maintains a balanced AVL tree. The tree consists of `TAVLTreeNode` (87) 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` (87) property.

See also: `TAVLTreeNode` (87)

### 2.3.2 Method overview

Page	Property	Description
83	Add	Add a new node to the tree
84	Clear	Clears the tree
85	ConsistencyCheck	Check the consistency of the tree
86	Create	Create a new instance of TAVLTree
83	Delete	Delete a node from the tree
86	Destroy	Destroy the TAVLTree instance
79	Find	Find a data item in the tree.
81	FindHighest	Find the highest (rightmost) node in the tree.
80	FindKey	Find a data item in the tree using alternate compare mechanism
81	FindLeftMost	Find the node most left to a specified data node
82	FindLeftMostKey	Find the node most left to a specified key node
82	FindLeftMostSameKey	Find the node most left to a specified node with the same data
80	FindLowest	Find the lowest (leftmost) node in the tree.
81	FindNearest	Find the node closest to the data in the tree
81	FindPointer	Search for a data pointer
80	FindPrecessor	
82	FindRightMost	Find the node most right to a specified node
82	FindRightMostKey	Find the node most right to a specified key node
83	FindRightMostSameKey	Find the node most right of a specified node with the same data
80	FindSuccessor	Find successor to node
85	FreeAndClear	Clears the tree and frees nodes
85	FreeAndDelete	Delete a node from the tree and destroy it
87	GetEnumerator	Get an enumerator for the tree.
84	MoveDataLeftMost	Move data to the nearest left element
84	MoveDataRightMost	Move data to the nearest right element
83	Remove	Remove a data item from the list.
84	RemovePointer	Remove a pointer item from the list.
86	ReportAsString	Return the tree report as a string
86	SetNodeManager	Set the node instance manager to use
85	WriteReportToStream	Write the contents of the tree consistency check to the stream

### 2.3.3 Property overview

Page	Property	Access	Description
87	Count	r	Number of nodes in the tree.
87	OnCompare	rw	Compare function used when comparing nodes

### 2.3.4 TAVLTree.Find

**Synopsis:** Find a data item in the tree.

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

**Visibility:** public

**Description:** Find uses the default OnCompare (87) 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 (81).

See also: [OnCompare \(87\)](#), [FindKey \(80\)](#)

### 2.3.5 TAVLTree.FindKey

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

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

**Visibility:** public

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

See also: [OnCompare \(87\)](#), [Find \(79\)](#)

### 2.3.6 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 \(80\)](#), [TAVLTree.MoveDataRightMost \(84\)](#)

### 2.3.7 TAVLTree.FindPrecessor

**Synopsis:**

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

**Visibility:** public

**Description:** FindPrecessor returns the predecessor 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 \(80\)](#), [TAVLTree.MoveDataLeftMost \(84\)](#)

### 2.3.8 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 \(81\)](#)

### 2.3.9 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 (80)

### 2.3.10 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 (81), FindLowest (80), Find (79), FindKey (80)

### 2.3.11 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 (79), where a custom compare function can be used.

The default OnCompare (87) compares the data pointers, so the default Find will return the same node as FindPointer

**See also:** TAVLTree.Find (79), TAVLTree.FindKey (80)

### 2.3.12 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 (82), TAVLTree.FindLeftMostKey (82), TAVLTree.FindRightMostKey (82)

### 2.3.13 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 (81), TAVLTree.FindLeftMostKey (82), TAVLTree.FindRightMostKey (82)

### 2.3.14 TAVLTree.FindLeftMostKey

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

**Declaration:** function FindLeftMostKey (Key: Pointer;  
                          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 (81), TAVLTree.FindRightMost (82), TAVLTree.FindRightMostKey (82)

### 2.3.15 TAVLTree.FindRightMostKey

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

**Declaration:** function FindRightMostKey (Key: Pointer;  
                          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 (81), TAVLTree.FindRightMost (82), TAVLTree.FindLeftMostKey (82)

### 2.3.16 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 (81), TAVLTree.FindLeftMostKey (82), TAVLTree.FindRightMostSameKey (83)

### 2.3.17 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 (82), TAVLTree.FindRightMostKey (82), TAVLTree.FindLeftMostSameKey (82)

### 2.3.18 TAVLTree.Add

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

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

**Visibility:** public

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

**See also:** TAVLTree.Delete (83), TAVLTree.Remove (83)

### 2.3.19 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 (89) 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 (83), TAVLTree.RemovePointer (84), TAVLTree.Clear (84)

### 2.3.20 TAVLTree.Remove

**Synopsis:** Remove a data item from the list.

**Declaration:** procedure Remove (Data: Pointer)

**Visibility:** public

**Description:** Remove finds the node associated with Data using find (79) and, if found, deletes it from the tree.  
Only the first occurrence of Data will be removed.

**See also:** TAVLTree.Delete (83), TAVLTree.RemovePointer (84), TAVLTree.Clear (84), TAVLTree.Find (79)

### 2.3.21 TAVLTree.RemovePointer

**Synopsis:** Remove a pointer item from the list.

**Declaration:** procedure RemovePointer(Data: Pointer)

**Visibility:** public

**Description:** Remove uses FindPointer (81) 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 (83), TAVLTree.Delete (83), TAVLTree.Clear (84)

### 2.3.22 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 (84)

### 2.3.23 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 (84)

### 2.3.24 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 (85) instead.

**See also:** TAVLTree.FreeAndClear (85), TAVLTree.Delete (83)

### 2.3.25 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 (84), TAVLTree.Delete (83), TAVLTree.FreeAndDelete (85)

### 2.3.26 TAVLTree.FreeAndDelete

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

**Declaration:** procedure FreeAndDelete (ANode: TAVLTreeNode)

**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 (84), TAVLTree.Delete (83), TAVLTree.FreeAndClear (85)

### 2.3.27 TAVLTree.ConsistencyCheck

**Synopsis:** Check the consistency of the tree

**Declaration:** function ConsistencyCheck : Integer

**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.

- 1The Count property doesn't match the actual node count
- 2A left node does not point to the correct parent
- 3A left node is larger than parent node
- 4A right node does not point to the correct parent
- 5A right node is less than parent node
- 6The balance of a node is not calculated correctly

See also: TAVLTree.WriteReportToStream (85)

### 2.3.28 TAVLTree.WriteReportToStream

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

**Declaration:** procedure WriteReportToStream(s: TStream; var StreamSize: Int64)

**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 (85)

### 2.3.29 TAVLTree.ReportAsString

**Synopsis:** Return the tree report as a string

**Declaration:** function ReportAsString : string

**Visibility:** public

**Description:** ReportAsString calls WriteReportToStream (85) and retuns the stream data as a string.

**See also:** TAVLTree.WriteReportToStream (85)

### 2.3.30 TAVLTree.SetNodeManager

**Synopsis:** Set the node instance manager to use

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

**Visibility:** public

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

By default, a single instance of TAVLTreeNodeMemManager (89) is used to manage the nodes of all TAVLTree instances.

**See also:** TBaseAVLTreeNodeManager (91), TAVLTreeNodeMemManager (89)

### 2.3.31 TAVLTree.Create

**Synopsis:** Create a new instance of TAVLTree

**Declaration:** constructor Create (OnCompareMethod: TListSortCompare)  
constructor Create

**Visibility:** public

**Description:** Create initializes a new instance of TAVLTree (78). An alternate OnCompare (87) can be provided: the default OnCompare method compares the 2 data pointers of a node.

**See also:** OnCompare (87)

### 2.3.32 TAVLTree.Destroy

**Synopsis:** Destroy the TAVLTree instance

**Declaration:** destructor Destroy; Override

**Visibility:** public

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

**See also:** TAVLTree.Create (86), TAVLTree.Clean (78)

### **2.3.33 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 ([88](#)).

**See also:** TAVLTreeNodeEnumerator ([88](#))

### **2.3.34 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 ([86](#))

### **2.3.35 TAVLTree.Count**

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

**Declaration:** Property Count : Integer

**Visibility:** public

**Access:** Read

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

## **2.4 TAVLTreeNode**

### **2.4.1 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 ([78](#)), TAVLTreeNode.Data (??)

### **2.4.2 Method overview**

Page	Property	Description
<a href="#">88</a>	Clear	Clears the node's data
<a href="#">88</a>	TreeDepth	Level of the node in the tree below

### **2.4.3 TAVLTreeNode.Clear**

**Synopsis:** Clears the node's data

**Declaration:** procedure Clear

**Visibility:** public

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

### **2.4.4 TAVLTreeNode.TreeDepth**

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

**Declaration:** function TreeDepth : Integer

**Visibility:** public

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

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

## **2.5 TAVLTreeNodeEnumerator**

### **2.5.1 Description**

TAVLTreeNodeEnumerator is a class which implements the enumerator interface for the AVL-Tree (78). It enumerates all the nodes in the tree.

**See also:** TAVLTree (78)

### **2.5.2 Method overview**

Page	Property	Description
88	Create	Create a new instance of TAVLTreeNodeEnumerator
89	MoveNext	Move to next node in the tree.

### **2.5.3 Property overview**

Page	Property	Access	Description
89	Current	r	Current node in the tree

### **2.5.4 TAVLTreeNodeEnumerator.Create**

**Synopsis:** Create a new instance of TAVLTreeNodeEnumerator

**Declaration:** constructor Create(Tree: TAVLTree)

**Visibility:** public

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

### **2.5.5 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 (80).

See also: TAVLTree.FindSuccessor (80)

### **2.5.6 TAVLTreeNodeEnumerator.Current**

Synopsis: Current node in the tree

Declaration: Property Current : TAVLTreeNode

Visibility: public

Access: Read

Description: Current is the current node in the enumeration.

See also: TAVLTreeNodeEnumerator.MoveNext (89)

## **2.6 TAVLTreeNodeMemManager**

### **2.6.1 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 (87), TAVLTree (78)

### **2.6.2 Method overview**

Page	Property	Description
90	Clear	Frees all unused nodes
90	Create	Create a new instance of TAVLTreeNodeMemManager
90	Destroy	
90	DisposeNode	Return a node to the free list
90	NewNode	Create a new TAVLTreeNode instance

### **2.6.3 Property overview**

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

## 2.6.4 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 (90)

## 2.6.5 TAVLTreeNodeMemManager.NewNode

**Synopsis:** Create a new TAVLTreeNode instance

**Declaration:** function NewNode : TAVLTreeNode;   Override

**Visibility:** public

**Description:** NewNode returns a new TAVLTreeNode (87) instance. If there is a node in the free list, it are returned. If no more free nodes are present, a new node is created.

See also: TAVLTreeNodeMemManager.DisposeNode (90)

## 2.6.6 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 (91), TAVLTreeNodeMemManager.MaximumFreeNodeRatio (91)

## 2.6.7 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 (90)

## 2.6.8 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 (90)

### 2.6.9 TAVLTreeNodeMemManager.MinimumFreeNode

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

**Declaration:** Property MinimumFreeNode : Integer

**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 (91)

### 2.6.10 TAVLTreeNodeMemManager.MaximumFreeNodeRatio

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

**Declaration:** Property MaximumFreeNodeRatio : Integer

**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 (91), TAVLTreeNodeMemManager.MinimumFreeNode (91)

### 2.6.11 TAVLTreeNodeMemManager.Count

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

**Declaration:** Property Count : Integer

**Visibility:** public

**Access:** Read

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

See also: TAVLTreeNodeMemManager.MinimumFreeNode (91), TAVLTreeNodeMemManager.MaximumFreeNodeRatio (91)

## 2.7 TBaseAVLTreeNodeManager

### 2.7.1 Description

TBaseAVLTreeNodeManager is an abstract class from which a descendent can be created that manages creating and disposing of tree nodes (instances of TAVLTreeNode (87)) for a TAVLTree (78) 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 (89).

The TAVLTree.SetNodeManager (86) method can be used to set the node manager that a TAVLTree instance should use.

See also: TAVLTreeNodeMemManager (89), TAVLTree.SetNodeManager (86), TAVLTreeNode (87)

### **2.7.2 Method overview**

Page	Property	Description
<a href="#">92</a>	DisposeNode	Called when the AVL tree no longer needs node
<a href="#">92</a>	NewNode	Called when the AVL tree needs a new node

### **2.7.3 TBaseAVLTreeNodeManager.DisposeNode**

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

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

**Visibility:** public

**Description:** DisposeNode is called by TAVLTree ([78](#)) when it no longer needs a TAVLTreeNode ([87](#)) instance.  
The manager may decide to re-use the instance for later use instead of destroying it.

**See also:** TBaseAVLTreeNodeManager.NewNode ([92](#)), TAVLTree.Delete ([83](#)), TAVLTreeNode ([87](#))

### **2.7.4 TBaseAVLTreeNodeManager.NewNode**

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

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

**Visibility:** public

**Description:** NewNode is called by TAVLTree ([78](#)) when it needs a new node in TAVLTree.Add ([83](#)). It must be implemented by descendants to return a new TAVLTreeNode ([87](#)) instance.

**See also:** TBaseAVLTreeNodeManager.DisposeNode ([92](#)), TAVLTree.Add ([83](#)), TAVLTreeNode ([87](#))

# Chapter 3

## Reference for unit 'base64'

### 3.1 Used units

Table 3.1: Used units by unit 'base64'

Name	Page
Classes	??
System	??
sysutils	??

### 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 ([97](#)) for encoding, and TBase64DecodingStream ([94](#)) 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 3.2: Enumeration values for type TBase64DecodingMode

Value	Explanation
bdmMIME	MIME encoding
bdmStrict	Strict encoding

TBase64DecodingMode determines the decoding algorithm used by TBase64DecodingStream (94). 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 (94) 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:** 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 (94) to do this. The Strict parameter is passed on to the constructor as bdmStrict or bdmMIME

**See also:** DecodeStringBase64 (94), TBase64DecodingStream (94)

### 3.4.2 EncodeStringBase64

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

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

**Visibility:** default

**Description:** EncodeStringBase64 encodes the string s using Base 64 encoding and returns the result. It uses a TBase64EncodingStream (97) to do this.

**See also:** DecodeStringBase64 (94), TBase64EncodingStream (97)

## 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 \(97\)](#)

### 3.6.2 Method overview

Page	Property	Description
<a href="#">95</a>	Create	Create a new instance of the TBase64DecodingStream class
<a href="#">95</a>	Read	Read and decrypt data from the source stream
<a href="#">95</a>	Reset	Reset the stream
<a href="#">96</a>	Seek	Set stream position.

### 3.6.3 Property overview

Page	Property	Access	Description
<a href="#">96</a>	EOF	r	
<a href="#">96</a>	Mode	rw	Decoding mode

### 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.Create \(97\)](#), [TBase64DecodingMode \(93\)](#)

### 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 \(96\)](#), [TBase64DecodingStream.Read \(95\)](#)

### 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.Write \(94\)](#), [TBase64DecodingStream.Seek \(96\)](#), [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 \(95\)](#), [TBase64DecodingStream.Write \(94\)](#), [TBase64EncodingStream.Seek \(98\)](#), [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 \(94\)](#)

## 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 yet contain all output: input must be a multiple of 3. In order to be sure that the output contains all encoded bytes, the Flush (97) 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 (94)

### 3.7.2 Method overview

Page	Property	Description
97	Destroy	Remove a TBase64EncodingStream instance from memory
97	Flush	Flush the remaining bytes to the output stream.
98	Seek	Position the stream
98	Write	Write data to the stream.

### 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.

See also: TBase64EncodingStream.Create (97)

### 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 (97)

### 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 ([98](#)), TBase64EncodingStream.Read ([97](#)), TBase64DecodingStream.Write ([94](#)), 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 it don't change the current file pointer position. The encryption stream is not seekable.

**Errors:** An EStreamError error is raised.

**See also:** TBase64EncodingStream.Read ([97](#)), TBase64EncodingStream.Write ([98](#)), TStream.Seek ([??](#))

# Chapter 4

## Reference for unit 'BlowFish'

### 4.1 Used units

Table 4.1: Used units by unit 'BlowFish'

Name	Page
Classes	??
System	??
sysutils	??

### 4.2 Overview

The BlowFish implements a class TBlowFish (100) to handle blowfish encryption/decryption of memory buffers, and 2 TStream (??) descendants TBlowFishDeCryptStream (101) which decrypts any data that is read from it on the fly, as well as TBlowFishEnCryptStream (102) 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 (100) array.

TBFBlock = Array[0..1] of LongInt

`TBFBlock` is the basic data structure used by the encrypting/decrypting routines in `TBlowFish` (100), `TBlowFishDeCryptStream` (101) and `TBlowFishEnCryptStream` (102). It is the basic encryption/decryption block for all encrypting/decrypting: all encrypting/decrypting happens on a `TBFBlock` structure.

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

`TBlowFishKey` is a data structure which keeps the encryption or decryption key for the `TBlowFish` (100), `TBlowFishDeCryptStream` (101) and `TBlowFishEnCryptStream` (102) 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` (104), `TBlowFishEncryptStream` (102) and `TBlowFishDecryptStream` (101) classes to report errors.

See also: `TBlowFishStream` (104), `TBlowFishEncryptStream` (102), `TBlowFishDecryptStream` (101)

## 4.5 TBlowFish

### 4.5.1 Description

`TBlowFish` is a simple class that can be used to encrypt/decrypt a single `TBFBlock` (100) data block with the `Encrypt` (101) and `Decrypt` (101) calls. It is used internally by the `TBlowFishEnCryptStream` (102) and `TBlowFishDeCryptStream` (101) classes to encrypt or decrypt the actual data.

See also: `TBlowFishEnCryptStream` (102), `TBlowFishDeCryptStream` (101)

### 4.5.2 Method overview

Page	Property	Description
100	Create	Create a new instance of the <code>TBlowFish</code> class
101	Decrypt	Decrypt a block
101	Encrypt	Encrypt a block

### 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` (101) and `Decrypt` (101).

See also: `Encrypt` (101), `Decrypt` (101)

#### **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 (100) specified when the TBlowFish instance was created.

**See also:** TBlowFishKey (100), Decrypt (101), Create (100)

#### **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 (100) specified when the TBlowFish instance was created. The data must have been encrypted with the same key and the Encrypt (101) call.

**See also:** TBlowFishKey (100), Encrypt (101), Create (100)

### **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 (102) stream must be used.

**See also:** Create (104), TBlowFishEncryptStream (102)

#### **4.6.2 Method overview**

Page	Property	Description
101	Read	Read data from the stream
102	Seek	Set the stream position.

#### **4.6.3 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 (104), TBlowFishEncryptStream (102)

#### 4.6.4 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 (100) exception will be raised.

**See also:** Read (101), EBlowFishError (100)

### 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 (101) stream must be used.

**See also:** Create (104), TBlowFishDecryptStream (101)

#### 4.7.2 Method overview

Page	Property	Description
102	Destroy	Free the TBlowFishEncryptStream
103	Flush	Flush the encryption buffer
103	Seek	Set the position in the stream
103	Write	Write data to the stream

#### 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 ([103](#)), Create ([104](#))

#### **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 ([103](#)) mechanism can be used to write the remaining bytes.

See also: TBlowFishEncryptStream.Read ([102](#))

#### **4.7.5 TBlowFishEncryptStream.Seek**

**Synopsis:** Set the position in the stream

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

**Visibility:** public

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

**Errors:** Calling this function always results in an EBlowFishError ([100](#)) exception.

See also: TBlowFishEncryptStream.Write ([103](#))

#### **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 ([103](#)), TBFBBlock ([100](#))

## 4.8 TBlowFishStream

### 4.8.1 Description

TBlowFishStream is an abstract class which is used as a parent class for TBlowFishEncryptStream (102) and TBlowFishDecryptStream (101). It simply provides a constructor and storage for a TBlowFish (100) 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 (102), TBlowFishDecryptStream (101), TBlowFish (100)

### 4.8.2 Method overview

Page	Property	Description
104	Create	Create a new instance of the TBlowFishStream class
104	Destroy	Destroy the TBlowFishStream instance.

### 4.8.3 Property overview

Page	Property	Access	Description
105	BlowFish	r	Blowfish instance used when encrypting/decrypting

### 4.8.4 TBlowFishStream.Create

Synopsis: Create a new instance of the TBlowFishStream class

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

Visibility: public

Description: Create initializes a new instance of TBlowFishStream, and creates an internal instance of TBlowFish (100) 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 (102), TBlowFishDecryptStream (101), TBlowFish (100)

### 4.8.5 TBlowFishStream.Destroy

Synopsis: Destroy the TBlowFishStream instance.

Declaration: destructor Destroy; Override

Visibility: public

Description: Destroy cleans up the internal TBlowFish (100) instance.

See also: Create (104), TBlowFish (100)

#### **4.8.6 TBlowFishStream.BlowFish**

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

**Declaration:** Property BlowFish : TBlowFish

**Visibility:** public

**Access:** Read

**Description:** BlowFish is the TBlowFish (100) 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 (102) and TBlowFishDecryptStream (101).

**See also:** TBlowFishEncryptStream (102), TBlowFishDecryptStream (101), TBlowFish (100)

# Chapter 5

## Reference for unit 'bufstream'

### 5.1 Used units

Table 5.1: Used units by unit 'bufstream'

Name	Page
Classes	??
System	??
sysutils	??

### 5.2 Overview

BufStream implements two one-way 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). The buffer size can be specified at creation time. 2 streams are implemented: TReadBufStream (109) which is for reading only, and TWriteBufStream (110) which is for writing only.

Buffered streams can help in speeding up read or write operations, especially when a lot of small read/write operations are done: it avoids doing a lot of operating system calls.

### 5.3 Constants, types and variables

#### 5.3.1 Constants

```
DefaultBufferCapacity : Integer = 16
```

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

### 5.4 TBufStream

#### 5.4.1 Description

TBufStream is the common ancestor for the TReadBufStream (109) and TWriteBufStream (110) streams. It completely handles the buffer memory management and position management. An in-

stance of `TBufStream` should never be created directly. It also keeps the instance of the source stream.

See also: [TReadBufStream \(109\)](#), [TWriteBufStream \(110\)](#)

#### 5.4.2 Method overview

Page	Property	Description
<a href="#">107</a>	Create	Create a new <code>TBufStream</code> instance.
<a href="#">107</a>	Destroy	Destroys the <code>TBufStream</code> instance

#### 5.4.3 Property overview

Page	Property	Access	Description
<a href="#">108</a>	Buffer	r	The current buffer
<a href="#">108</a>	BufferPos	r	Current buffer position.
<a href="#">108</a>	BufferSize	r	Amount of data in the buffer
<a href="#">108</a>	Capacity	rw	Current buffer capacity

#### 5.4.4 `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` ([106](#)) is used as the capacity.

An instance of `TBufStream` should never be instantiated directly. Instead, an instance of `TReadBufStream` ([109](#)) or `TWriteBufStream` ([110](#)) should be created.

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

See also: `TBufStream.Destroy` ([107](#)), `TReadBufStream` ([109](#)), `TWriteBufStream` ([110](#))

#### 5.4.5 `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` ([107](#)), `TReadBufStream` ([109](#)), `TWriteBufStream` ([110](#))

#### 5.4.6 **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 \(107\)](#), [TBufStream.Capacity \(108\)](#), [TBufStream.BufferSize \(108\)](#)

#### 5.4.7 **TBufStream.Capacity**

**Synopsis:** Current buffer capacity

**Declaration:** Property Capacity : Integer

**Visibility:** public

**Access:** Read,Write

**Description:** Capacity is the amount of memory the buffer occupies. To change the buffer size, the capacity can be set. Note that the capacity cannot be set to a value that is less than the current buffer size, i.e. the current amount of data in the buffer.

See also: [TBufStream.Create \(107\)](#), [TBufStream.Buffer \(108\)](#), [TBufStream.BufferSize \(108\)](#), [TBufStream.BufferPos \(108\)](#)

#### 5.4.8 **TBufStream.BufferPos**

**Synopsis:** Current buffer position.

**Declaration:** Property BufferPos : Integer

**Visibility:** public

**Access:** Read

**Description:** BufPos is the current stream position in the buffer. Depending on whether the stream is used for reading or writing, data will be read from this position, or will be written at this position in the buffer.

See also: [TBufStream.Create \(107\)](#), [TBufStream.Buffer \(108\)](#), [TBufStream.BufferSize \(108\)](#), [TBufStream.Capacity \(108\)](#)

#### 5.4.9 **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 (108).

See also: [TBufStream.Create \(107\)](#), [TBufStream.Buffer \(108\)](#), [TBufStream.BufferPos \(108\)](#), [TBufStream.Capacity \(108\)](#)

## 5.5 TReadBufStream

### 5.5.1 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 \(106\)](#), [TWriteBufStream \(110\)](#)

### 5.5.2 Method overview

Page	Property	Description
<a href="#">109</a>	Read	Reads data from the stream
<a href="#">109</a>	Seek	Set location in the buffer

### 5.5.3 TReadBufStream.Seek

Synopsis: Set location in the buffer

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

Visibility: public

Description: Seek sets the location in the buffer. Currently, only a forward seek is allowed. It is emulated by reading and discarding data. For an explanation of the parameters, see [TStream.Seek" \(??\)](#)

The seek method needs enhancement to enable it to do a full-featured seek. This may be implemented in a future release of Free Pascal.

Errors: In case an illegal seek operation is attempted, an exception is raised.

See also: [TWriteBufStream.Seek \(110\)](#), [TReadBufStream.Read \(109\)](#), [TReadBufStream.Write \(109\)](#)

### 5.5.4 TReadBufStream.Read

Synopsis: Reads data from the stream

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

Visibility: public

Description: Read reads at most ACount bytes from the stream and places them in Buffer. The number of actually read bytes is returned.

TReadBufStream first reads whatever data is still available in the buffer, and then refills the buffer, after which it continues to read data from the buffer. This is repeated until ACount bytes are read, or no more data is available.

See also: [TReadBufStream.Seek \(109\)](#), [TReadBufStream.Read \(109\)](#)

## 5.6 TWriteBufStream

### 5.6.1 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 \(106\)](#), [TReadBufStream \(109\)](#)

### 5.6.2 Method overview

Page	Property	Description
<a href="#">110</a>	Destroy	Remove the TWriteBufStream instance from memory
<a href="#">110</a>	Seek	Set stream position.
<a href="#">110</a>	Write	Write data to the stream

### 5.6.3 TWriteBufStream.Destroy

Synopsis: Remove the TWriteBufStream instance from memory

Declaration: `destructor Destroy; Override`

Visibility: public

Description: `Destroy` flushes the buffer and then calls the inherited `Destroy (107)`.

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

See also: [Create \(107\)](#), [TBufStream.Destroy \(107\)](#)

### 5.6.4 TWriteBufStream.Seek

Synopsis: Set stream position.

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

Visibility: public

Description: `Seek` always raises an `EStreamError` exception, except when the seek operation would not alter the current position.

A later implementation may perform a proper seek operation by flushing the buffer and doing a seek on the source stream.

See also: [TWriteBufStream.Write \(110\)](#), [TWriteBufStream.Read \(110\)](#), [TReadBufStream.Seek \(109\)](#)

### 5.6.5 TWriteBufStream.Write

Synopsis: Write data to the stream

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

Visibility: public

**Description:** Write writes at most ACount bytes from ABuffer to the stream. The data is written to the internal buffer first. As soon as the internal buffer is full, it is flushed to the destination stream, and the internal buffer is filled again. This process continues till all data is written (or an error occurs).

**Errors:** An exception may occur if the destination stream has problems writing.

**See also:** [TWriteBufStream.Seek \(110\)](#), [TWriteBufStream.Read \(110\)](#), [TReadBufStream.Write \(109\)](#)

# Chapter 6

## Reference for unit 'CacheCls'

### 6.1 Used units

Table 6.1: Used units by unit 'CacheCls'

Name	Page
System	??
sysutils	??

### 6.2 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 called TCache

### 6.3 Constants, types and variables

#### 6.3.1 Resource strings

SInvalidIndex = 'Invalid index %i'

Message shown when an invalid index is passed.

#### 6.3.2 Types

PCacheSlot = ^TCacheSlot

Pointer to TCacheSlot ([113](#)) record.

PCacheSlotArray = ^TCacheSlotArray

Pointer to TCacheSlotArray ([113](#)) array

TCacheSlot = record

```
Prev : PCacheSlot;
Next : PCacheSlot;
Data : Pointer;
Index : Integer;
end
```

`TCacheSlot` is internally used by the `TCache` (113) class. It represents 1 element in the linked list.

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

`TCacheSlotArray` is an array of `TCacheSlot` items. Do not use `TCacheSlotArray` directly, instead, use `PCacheSlotArray` (112) 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` (114) call to determine whether the item to be added is a new item or not. The function returns `True` if the 2 data pointers `AData1` and `AData2` should be considered equal, or `False` when they are not.

For most purposes, comparing the pointers will be enough, but if the pointers are `ansistrings`, then the contents should be compared.

## 6.4 ECacheError

### 6.4.1 Description

Exception class used in the `cachecls` unit.

## 6.5 TCache

### 6.5.1 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` (116) property offers indexed access to the array of items. When accessing the array through this property, the `MRUSlot` (116) property is updated.

### 6.5.2 Method overview

Page	Property	Description
<a href="#">114</a>	Add	Add a data element to the list.
<a href="#">115</a>	AddNew	Add a new item to the list.
<a href="#">114</a>	Create	Create a new cache class.
<a href="#">114</a>	Destroy	Free the TCache class from memory
<a href="#">115</a>	FindSlot	Find data pointer in the list
<a href="#">115</a>	IndexOf	Return index of a data pointer in the list.
<a href="#">116</a>	Remove	Remove a data item from the list.

### 6.5.3 Property overview

Page	Property	Access	Description
<a href="#">116</a>	Data	rw	Indexed access to data items
<a href="#">117</a>	LRUSlot	r	Last used item
<a href="#">116</a>	MRUSlot	rw	Most recent item slot.
<a href="#">118</a>	OnFreeSlot	rw	Event called when a slot is freed
<a href="#">117</a>	OnIsDataEqual	rw	Event to compare 2 items.
<a href="#">117</a>	SlotCount	rw	Number of slots in the list
<a href="#">117</a>	Slots	r	Indexed array to the slots

### 6.5.4 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 ([117](#))

### 6.5.5 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 ([114](#))

### 6.5.6 TCache.Add

Synopsis: Add a data element to the list.

Declaration: function Add (AData: Pointer) : Integer

Visibility: public

**Description:** Add checks whether AData is already in the list. If so, the item is added to the top of the MRU list.

If the item is not yet in the list, then the item is added to the list and placed at the top of the MRU list using the AddNew ([115](#)) 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 ([115](#)), TCache.FindSlot ([115](#)), TCache.IndexOf ([115](#)), TCache.Data ([116](#)), TCache.MRUSlot ([116](#))

### **6.5.7 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 ([114](#)) 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 ([114](#)), TCache.FindSlot ([115](#)), TCache.IndexOf ([115](#)), TCache.Data ([116](#)), TCache.MRUSlot ([116](#))

### **6.5.8 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 ([117](#)) event must be set.

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

**See also:** TCache.IndexOf ([115](#)), TCache.Add ([114](#)), TCache.OnIsDataEqual ([117](#))

### **6.5.9 TCache.IndexOf**

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

**Declaration:** function IndexOf(AData: Pointer) : Integer

**Visibility:** public

**Description:** IndexOF searches in the list for a slot with data pointer that matches AData and returns the index of the slot.

If no item with data pointer that matches AData is found, -1 is returned.

For this function to work correctly, the OnIsDataEqual ([117](#)) event must be set.

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

**See also:** TCache.FindSlot ([115](#)), TCache.Add ([114](#)), TCache.OnIsDataEqual ([117](#))

### 6.5.10 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 ([115](#))

### 6.5.11 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 ([116](#)) will point to the accessed item. The access is both read and write.

The index is zero-based and can maximally be SlotCount-1 ([117](#)). Providing an invalid index will result in an exception.

**See also:** TCache.MRUSlot ([116](#))

### 6.5.12 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 ([116](#)) property, or when an item is added to the list with Add ([114](#)) or AddNew ([115](#))

**See also:** TCache.Add ([114](#)), TCache.AddNew ([115](#)), TCache.Data ([116](#)), TCache.LRUSlot ([117](#))

### 6.5.13 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 ([114](#)), TCache.AddNew ([115](#)), TCache.Data ([116](#)), TCache.MRUSlot ([116](#))

### 6.5.14 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 ([118](#)) is called.

**See also:** TCache.Create ([114](#)), TCache.Data ([116](#)), TCache.Slots ([117](#))

### 6.5.15 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 PCacheSlot 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 ([117](#)). Providing an invalid index will result in an exception.

**See also:** TCache.Data ([116](#)), TCache.SlotCount ([117](#))

### 6.5.16 TCache.OnIsDataEqual

**Synopsis:** Event to compare 2 items.

**Declaration:** Property OnIsDataEqual : TOnIsDataEqual

**Visibility:** public

**Access:** Read,Write

**Description:** `OnIsDataEqual` is used by `FindSlot` (115) and `IndexOf` (115) to compare items when looking for a particular item. These functions are called by the `Add` (114) 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` (115), `TCache.IndexOf` (115), `TCache.Add` (114)

### 6.5.17 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` (114), `TCache.AddNew` (115), `TCache.SlotCount` (117)

# Chapter 7

## Reference for unit 'contnrs'

### 7.1 Used units

Table 7.1: Used units by unit 'contnrs'

Name	Page
Classes	??
System	??
sysutils	??

### 7.2 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 of 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.

### 7.3 Constants, types and variables

#### 7.3.1 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.

### 7.3.2 Types

```
PBucket = ^TBucket
```

Pointer to TBucket ([120](#))" type.

```
PHashItem = ^THashItem
```

PHashItem is a pointer type, pointing to the THashItem ([122](#)) record.

```
PHashItemList = ^THashItemList
```

PHashItemList is a pointer to the THashItemList ([122](#)). It's used in the TFPHashList ([139](#)) as a pointer to the memory area containing the hash item records.

```
PHashTable = ^THashTable
```

PHashTable is a pointer to the THashTable ([122](#)). It's used in the TFPHashList ([139](#)) as a pointer to the memory area containing the hash values.

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

TBucket describes 1 bucket in the TCustomBucketList ([130](#)) class. It is a container for TBucketItem ([121](#)) records. It should never be used directly.

```
TBucketArray = Array of TBucket
```

Array of TBucket ([120](#)) records.

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

`TBucketItem` is a record used for internal use in `TCustomBucketList` (130). It should not be necessary to use it directly.

```
TBucketItemArray = Array of TBucketItem
Array of TBucketItem records

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

Table 7.2: Enumeration values for type `TBucketListSizes`

Value	Explanation
bl128	List with 128 buckets
bl16	List with 16 buckets
bl2	List with 2 buckets
bl256	List with 256 buckets
bl32	List with 32 buckets
bl4	List with 4 buckets
bl64	List with 64 buckets
bl8	List with 8 buckets

`TBucketListSizes` is used to set the bucket list size: It specified the number of buckets created by `TBucketList` (123).

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

`TBucketProc` is the prototype for the `TCustomBucketList.Foreach` (132) 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` (121) prototype.

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

`TBucketProcObject` is the prototype for the `TCustomBucketList.Foreach` (132) 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` (121) prototype.

```
TDataIteratorMethod = procedure(Item: Pointer; const Key: string;
                                 var Continue: Boolean) of object
```

`TDataIteratorMethod` is a callback prototype for the `TFPDataHashTable.Iterate` (138) method. It is called for each data pointer in the hash list, passing the key (`key`) and data pointer (`item`) for each item in the list. If `Continue` is set to `false`, the iteration stops.

```
THashFunction = function(const S: string; const TableSize: LongWord
                         : LongWord
```

THashFunction is the prototype for a hash calculation function. It should calculate a hash of string S, where the hash table size is TableSize. The return value should be the hash value.

```
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.

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

THashItemList is an array type, primarily used to be able to define the PHashItemList ([120](#)) type. It's used in the TFPHashList ([139](#)) class.

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

THashTable defines an array of integers, used to hold hash values. It's mainly used to define the PHashTable ([120](#)) class.

```
THTCustomNodeClass = Class of THTCustomNode
```

THTCustomNodeClass was used by TFPCustomHashTable ([132](#)) 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 ([138](#)) method.

```
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 TFPObjecList.ForEachCall ([164](#)) 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)
```

---

TObjectListCallback is used as the prototype for the TFPObjectList.ForEachCall (164) 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.

```
TStringIteratorMethod = procedure(Item: string; const Key: string;
                                  var Continue: Boolean) of object
```

TStringIteratorMethod is the callback prototype for the Iterate (132) method. It is called for each element in the hash table, with the string. If Continue is set to false, the iteration stops.

## 7.4 Procedures and functions

### 7.4.1 RSHash

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

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

**Visibility:** default

**Description:** RSHash is the standard hash calculating function used in the TFPCustomHashTable (132) hash class. It's Robert Sedgwick's "Algorithms in C" hash function.

**Errors:** None.

See also: TFPCustomHashTable (132)

## 7.5 EDuplicate

### 7.5.1 Description

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

See also: TFPCustomHashTable.Add (132)

## 7.6 EKeyNotFound

### 7.6.1 Description

Exception raised when a key is not found.

See also: TFPCustomHashTable.Delete (135)

## 7.7 TBucketList

### 7.7.1 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 (130)

### 7.7.2 Method overview

Page	Property	Description
124	Create	Create a new TBucketList instance.

### 7.7.3 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 (121)

## 7.8 TClassList

### 7.8.1 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 (127), TObjectList (171)

### 7.8.2 Method overview

Page	Property	Description
125	Add	Add a new class pointer to the list.
125	Extract	Extract a class pointer from the list.
126	First	Return first non-nil class pointer
125	IndexOf	Search for a class pointer in the list.
126	Insert	Insert a new class pointer in the list.
126	Last	Return last non-Nil class pointer
125	Remove	Remove a class pointer from the list.

### 7.8.3 Property overview

Page	Property	Access	Description
126	Items	rw	Index based access to class pointers.

### 7.8.4 TClassList.Add

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

**Declaration:** function Add(AClass: TClass) : Integer

**Visibility:** public

**Description:** Add adds AClass to the list, and returns the position at which it was added. It simply overrides the TList (??) 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 (125), #rtl.classes.tlist.add (??)

### 7.8.5 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 (125), #rtl.classes.Tlist.Extract (??)

### 7.8.6 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 (125), #rtl.classes.Tlist.Remove (??)

### 7.8.7 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 (??)

### 7.8.8 TClassList.First

**Synopsis:** Return 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.

**Errors:** None.

**See also:** TClassList.Last ([126](#)), TClassList.Pack ([124](#))

### 7.8.9 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.

**Errors:** None.

**See also:** TClassList.First ([126](#)), TClassList.Pack ([124](#))

### 7.8.10 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 ([125](#)), TClassList.Remove ([125](#))

### 7.8.11 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 ([??](#))

## 7.9 TComponentList

### 7.9.1 Description

TComponentList is a TObjectList (171) 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 (171), 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 (??), TFPOObjectList (158), TObjectList (171), TClassList (124)

### 7.9.2 Method overview

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

### 7.9.3 Property overview

Page	Property	Access	Description
129	Items	rw	Index-based access to the elements in the list.

### 7.9.4 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 (171), #rtl.classes.TComponent (??)

### 7.9.5 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 \(172\)](#)

### 7.9.6 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 \(171\)](#) 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 \(128\)](#), [TObjectList.Extract \(172\)](#)

### 7.9.7 TComponentList.Remove

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

**Declaration:** function Remove (AComponent: TComponent) : Integer

**Visibility:** public

**Description:** Remove removes item from the list, and if the list owns its 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 \(171\)](#) so it only accepts TComponent descendants. It introduces no new behaviour.

**Errors:** None.

See also: [TComponentList.Extract \(128\)](#), [TObjectList.Remove \(173\)](#)

### 7.9.8 TComponentList.IndexOf

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

**Declaration:** function IndexOf (AComponent: TComponent) : Integer

**Visibility:** public

**Description:** IndexOf searches for an instance in the list and returns its 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 \(173\)](#)

### 7.9.9 TComponentList.First

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

**Declaration:** function First : TComponent

**Visibility:** public

**Description:** First overrides the implementation of it's ancestors to return the first non-nil instance of TComponent in the list. If no non-nil instance is found, Nil is returned.

**Errors:** None.

**See also:** TComponentList.Last ([129](#)), TObjectList.First ([174](#))

### 7.9.10 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 ([129](#)), TObjectList.Last ([174](#))

### 7.9.11 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 ([174](#)), TComponentList.Add ([127](#)), TComponentList.Remove ([128](#))

### 7.9.12 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 ([175](#))

## 7.10 TCustomBucketList

### 7.10.1 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 (123). 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 (130).

See also: TBucketList (123)

### 7.10.2 Method overview

Page	Property	Description
131	Add	Add an item to the list
131	Assign	Assign one bucket list to another
130	Clear	Clear the list
130	Destroy	Frees the bucketlist from memory
131	Exists	Check if an item exists in the list.
131	Find	Find an item in the list
132	ForEach	Loop over all items.
132	Remove	Remove an item from the list.

### 7.10.3 Property overview

Page	Property	Access	Description
132	Data	rw	Associative array for data pointers

### 7.10.4 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.

### 7.10.5 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 ([131](#))

### 7.10.6 TCustomBucketList.Add

**Synopsis:** Add an item to the list

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

**Visibility:** public

**Description:** Add adds AItem with it's associated AData to the list and returns AData.

**Errors:** If AItem is already in the list, an EListError exception will be raised.

**See also:** TCustomBucketList.Exists ([131](#)), TCustomBucketList.Clear ([130](#))

### 7.10.7 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 ([131](#)), TCustomBucketList.Clear ([130](#))

### 7.10.8 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 ([131](#)).

**See also:** TCustomBucketList.Find ([131](#))

### 7.10.9 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 ([131](#))

### 7.10.10 TCustomBucketList.ForEach

**Synopsis:** Loop over all items.

**Declaration:** function ForEach (AProc: TBucketProc; AInfo: Pointer) : Boolean  
function ForEach (AProc: TBucketProcObject) : Boolean

**Visibility:** public

**Description:** Fforeach 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 \(132\)](#)

### 7.10.11 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 \(131\)](#)

### 7.10.12 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 \(131\)](#), [TCustomBucketList.Exists \(131\)](#)

## 7.11 TFPCustomHashTable

### 7.11.1 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 ([123](#)).

A TFPHasList should be used when fast lookup of data based on some key is required. The other container objects only offer linear search methods, while the hash list offers faster search mechanisms.

See also: [THTCustomNode \(167\)](#), [TFPObjectList \(158\)](#), [RSHash \(123\)](#)

### 7.11.2 Method overview

Page	Property	Description
<a href="#">134</a>	ChangeTableSize	Change the table size of the hash table.
<a href="#">134</a>	Clear	Clear the hash table.
<a href="#">133</a>	Create	Instantiate a new <code>TFPCustomHashTable</code> instance using the default hash mechanism
<a href="#">134</a>	CreateWith	Instantiate a new <code>TFPCustomHashTable</code> instance with given algorithm and size
<a href="#">135</a>	Delete	Delete a key from the hash list.
<a href="#">134</a>	Destroy	Free the hash table.
<a href="#">135</a>	Find	Search for an item with a certain key value.
<a href="#">135</a>	IsEmpty	Check if the hash table is empty.

### 7.11.3 Property overview

Page	Property	Access	Description
<a href="#">137</a>	AVGChainLen	r	Average chain length
<a href="#">136</a>	Count	r	Number of items in the hash table.
<a href="#">138</a>	Density	r	Number of filled slots
<a href="#">135</a>	HashFunction	rw	Hash function currently in use
<a href="#">136</a>	HashTable	r	Hash table instance
<a href="#">136</a>	HashTableSize	rw	Size of the hash table
<a href="#">137</a>	LoadFactor	r	Fraction of count versus size
<a href="#">137</a>	MaxChainLength	r	Maximum chain length
<a href="#">137</a>	NumberOfCollisions	r	Number of extra items
<a href="#">136</a>	VoidSlots	r	Number of empty slots in the hash table.

### 7.11.4 `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 ([123](#))

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

See also: [CreateWith \(134\)](#)

### 7.11.5 **TFPCustomHashTable.CreateWith**

**Synopsis:** Instantiate a new TFPCustomHashTable instance with given algorithm and size

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

**Visibility:** public

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

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

**See also:** Create ([133](#))

### 7.11.6 **TFPCustomHashTable.Destroy**

**Synopsis:** Free the hash table.

**Declaration:** destructor Destroy; Override

**Visibility:** public

**Description:** Destroy removes the hash table from memory. If any data was associated with the keys in the hash table, then this data is not freed. This must be done by the programmer.

**Errors:** None.

**See also:** Destroy ([134](#)), Create ([133](#)), Create ([134](#)), THTCustomNode.Data ([167](#))

### 7.11.7 **TFPCustomHashTable.ChangeTableSize**

**Synopsis:** Change the table size of the hash table.

**Declaration:** procedure ChangeTableSize(const ANewSize: LongWord); Virtual

**Visibility:** public

**Description:** ChangeTableSize changes the size of the hash table: it recomputes the hash value for all of the keys in the table, so this is an expensive operation.

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

**See also:** HashTableSize ([136](#))

### 7.11.8 **TFPCustomHashTable.Clear**

**Synopsis:** Clear the hash table.

**Declaration:** procedure Clear; Virtual

**Visibility:** public

**Description:** Clear removes all keys and their associated data from the hash table. The data itself is not freed from memory, this should be done by the programmer.

**Errors:** None.

**See also:** Destroy ([134](#))

### 7.11.9 **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 ([135](#)), TFPCustomHashTable.Add ([132](#))

### 7.11.10 **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 ([167](#)) 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 ([135](#)) method should be used instead.

**Errors:** None.

**See also:** Add ([132](#)), Delete ([135](#))

### 7.11.11 **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 ([136](#)), TFPCustomHashTable.HashTableSize ([136](#)), TFPCustomHashTable.AVGChainLen ([137](#)), TFPCustomHashTable.MaxChainLength ([137](#))

### 7.11.12 **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 ([132](#)). Note that setting the hash function does NOT 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:** SetHashFunction ([132](#)), HashTableSize ([136](#))

### 7.11.13 **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 ([135](#)), TFPCustomHashTable.HashTableSize ([136](#)), TFPCustomHashTable.AVGChainLen ([137](#)), TFPCustomHashTable.MaxChainLength ([137](#))

### 7.11.14 **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 ([135](#)), TFPCustomHashTable.Count ([136](#)), TFPCustomHashTable.AVGChainLen ([137](#)), TFPCustomHashTable.MaxChainLength ([137](#)), TFPCustomHashTable.VoidSlots ([136](#)), TFPCustomHashTable.Density ([138](#))

### 7.11.15 **TFPCustomHashTable.HashTable**

**Synopsis:** Hash table instance

**Declaration:** Property HashTable : TFPOBJECTList

**Visibility:** public

**Access:** Read

**Description:** TFPCustomHashTable is the internal list object (TFPOBJECTList ([158](#)) used for the hash table. Each element in this table is again a TFPOBJECTList ([158](#)) instance or Nil.

### 7.11.16 **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 ([135](#)), TFPCustomHashTable.Count ([136](#)), TFPCustomHashTable.AVGChainLen ([137](#)), TFPCustomHashTable.MaxChainLength ([137](#)), TFPCustomHashTable.LoadFactor ([137](#)), TFPCustomHashTable.Density ([138](#)), TFPCustomHashTable.NumberOfCollisions ([137](#))

### 7.11.17 **TFPCustomHashTable.LoadFactor**

**Synopsis:** Fraction of count versus size

**Declaration:** Property LoadFactor : Double

**Visibility:** public

**Access:** Read

**Description:** LoadFactor is the ratio of elements in the table versus table size. Ideally, this should be as small as possible.

**See also:** TFPCustomHashTable.IsEmpty ([135](#)), TFPCustomHashTable.Count ([136](#)), TFPCustomHashTable.AVGChainLen ([137](#)), TFPCustomHashTable.MaxChainLength ([137](#)), TFPCustomHashTable.VoidSlots ([136](#)), TFP-CustomHashTable.Density ([138](#)), TFPCustomHashTable.NumberOfCollisions ([137](#))

### 7.11.18 **TFPCustomHashTable.AVGChainLen**

**Synopsis:** Average chain length

**Declaration:** Property AVGChainLen : Double

**Visibility:** public

**Access:** Read

**Description:** AVGChainLen is the average chain length, i.e. the ratio of elements in the table versus the number of filled slots. Calculating this is an expensive operation.

**See also:** TFPCustomHashTable.IsEmpty ([135](#)), TFPCustomHashTable.Count ([136](#)), TFPCustomHashTable.LoadFactor ([137](#)), TFPCustomHashTable.MaxChainLength ([137](#)), TFPCustomHashTable.VoidSlots ([136](#)), TFP-CustomHashTable.Density ([138](#)), TFPCustomHashTable.NumberOfCollisions ([137](#))

### 7.11.19 **TFPCustomHashTable.MaxChainLength**

**Synopsis:** Maximum chain length

**Declaration:** Property MaxChainLength : LongWord

**Visibility:** public

**Access:** Read

**Description:** MaxChainLength is the length of the longest chain in the hash table. Calculating this is an expensive operation.

**See also:** TFPCustomHashTable.IsEmpty ([135](#)), TFPCustomHashTable.Count ([136](#)), TFPCustomHashTable.LoadFactor ([137](#)), TFPCustomHashTable.AvgChainLength ([132](#)), TFPCustomHashTable.VoidSlots ([136](#)), TFP-CustomHashTable.Density ([138](#)), TFPCustomHashTable.NumberOfCollisions ([137](#))

### 7.11.20 **TFPCustomHashTable.NumberOfCollisions**

**Synopsis:** Number of extra items

**Declaration:** Property NumberOfCollisions : LongWord

**Visibility:** public

Access: Read

Description: NumberOfCollisions is the number of items which are not the first item in a chain. If this number is too big, the hash size may be too small.

See also: TFPCustomHashTable.IsEmpty ([135](#)), TFPCustomHashTable.Count ([136](#)), TFPCustomHashTable.LoadFactor ([137](#)), TFPCustomHashTable.AvgChainLength ([132](#)), TFPCustomHashTable.VoidSlots ([136](#)), TFPCustomHashTable.Density ([138](#))

### **7.11.21 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 ([135](#)), TFPCustomHashTable.Count ([136](#)), TFPCustomHashTable.LoadFactor ([137](#)), TFPCustomHashTable.AvgChainLength ([132](#)), TFPCustomHashTable.VoidSlots ([136](#)), TFPCustomHashTable.Density ([138](#))

## **7.12 TFPDataHashTable**

### **7.12.1 Description**

TFPDataHashTable is a TFPCustomHashTable ([132](#)) 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 ([156](#)), or for string data, TFPStringHashTable ([165](#)) is more suitable. The data pointers are exposed with their keys through the Items ([139](#)) property.

See also: TFPOBJECTHashTable ([156](#)), TFPStringHashTable ([165](#)), Items ([139](#))

### **7.12.2 Method overview**

Page	Property	Description
<a href="#">139</a>	Add	Add a data pointer to the list.
<a href="#">138</a>	Iterate	Iterate over the pointers in the hash table

### **7.12.3 Property overview**

Page	Property	Access	Description
<a href="#">139</a>	Items	rw	Key-based access to the items in the table

### **7.12.4 TFPDataHashTable.Iterate**

Synopsis: Iterate over the pointers in the hash table

Declaration: function Iterate(aMethod: TDataIteratorMethod) : 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.

See also: [ForeachCall \(119\)](#)

### 7.12.5 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 \(139\)](#)

### 7.12.6 TFPDataHashTable.Items

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

**Declaration:** `Property Items[index: string]: Pointer; default`

**Visibility:** public

**Access:** Read,Write

**Description:** `Items` provides access to the items in the hash table using their key: the array index `Index` is the key. A key which is not present will result in an `Nil` pointer.

See also: [TFPStringHashTable.Add \(166\)](#)

## 7.13 TFPHashList

### 7.13.1 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 pointers.

If a base class for an own hash class is wanted, the `TFPCustomHashTable` (132) class can be used. If a hash class for objects is needed instead of pointers, the `TFPHashObjectList` (149) class can be used.

See also: [TFPCustomHashTable \(132\)](#), [TFPHashObjectList \(149\)](#), [TFPDataHashTable \(138\)](#), [TFPStringHashTable \(165\)](#)

### 7.13.2 Method overview

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

### 7.13.3 Property overview

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

### 7.13.4 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 (140)

### 7.13.5 TFPHashList.Destroy

**Synopsis:** Removes an instance of the hashlist from the heap

**Declaration:** destructor Destroy; Override

**Visibility:** public

**Description:** Destroy cleans up the memory structures maintained by the hashlist and removes the TFPHashList instance from the heap.

Destroy should not be called directly, it's better to use Free or FreeAndNil instead.

See also: [TFPHashList.Create](#) (140), [TFPHashList.Clear](#) (141)

### 7.13.6 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](#) (143), [TFPHashList.Remove](#) (144), [TFPHashList.Delete](#) (142)

### 7.13.7 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](#) (143), [TFPHashList.Remove](#) (144), [TFPHashList.Delete](#) (142), [TFPHashList.Add](#) (141)

### 7.13.8 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](#) (141), [TFPHashList.Find](#) (143), [TFPHashList.FindIndexOf](#) (143), [TFPHashList.FindWithHash](#) (144)

### 7.13.9 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.HashOfName \(139\)](#), [TFPHashList.Find \(143\)](#), [TFPHashList.FindIndexOf \(143\)](#), [TFPHashList.FindWithHash \(144\)](#)

### 7.13.10 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

### 7.13.11 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 \(143\)](#)[TFPHashList.Remove \(144\)](#)[TFPHashList.Add \(141\)](#)

### 7.13.12 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.

### 7.13.13 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 \(141\)](#)

### 7.13.14 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 \(142\)](#), [TFPHashList.Remove \(144\)](#), [TFPHashList.Clear \(141\)](#)

### 7.13.15 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 \(141\)](#), [TFPHashList.NameOfIndex \(141\)](#), [TFPHashList.Find \(143\)](#), [TFPHashList.FindIndexOf \(143\)](#), [TFPHashList.FindWithHash \(144\)](#)

### 7.13.16 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 \(141\)](#), [TFPHashList.NameOfIndex \(141\)](#), [TFPHashList.IndexOf \(143\)](#), [TFPHashList.FindIndexOf \(143\)](#), [TFPHashList.FindWithHash \(144\)](#)

### 7.13.17 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.

**See also:** [TFPHashList.HashOfIndex \(141\)](#), [TFPHashList.NameOfIndex \(141\)](#), [TFPHashList.IndexOf \(143\)](#), [TFPHashList.Find \(143\)](#), [TFPHashList.FindWithHash \(144\)](#)

### 7.13.18 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 \(141\)](#), [TFPHashList.NameOfIndex \(141\)](#), [TFPHashList.IndexOf \(143\)](#),  
[TFPHashList.Find \(143\)](#), [TFPHashList.FindIndexOf \(143\)](#)

### 7.13.19 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.

### 7.13.20 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 \(142\)](#), [TFPHashList.Clear \(141\)](#), [TFPHashList.Extract \(143\)](#)

### 7.13.21 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 \(141\)](#)

### 7.13.22 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

### 7.13.23 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.

### 7.13.24 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 (145), Items (146)

### 7.13.25 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 (145), Items (146)

### 7.13.26 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 ([145](#)).

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

**See also:** Capacity ([145](#)), Count ([145](#))

### 7.13.27 TFPHashList.List

**Synopsis:** Low-level hash list

**Declaration:** Property List : PHashItemList

**Visibility:** public

**Access:** Read

**Description:** List exposes the low-level item list ([122](#)). It should not be used directly.

**See also:** Strs ([146](#)), THashItemList ([122](#))

### 7.13.28 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 ([146](#))

## 7.14 TFPHashObject

### 7.14.1 Description

TFPHashObject is a TObject descendent which is aware of the TFPHashObjectList ([149](#)) 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 ([148](#)), TFPHashObject.ChangeOwner ([147](#)), TFPHashObject.ChangeOwnerAndName ([148](#))

### 7.14.2 Method overview

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

### 7.14.3 Property overview

Page	Property	Access	Description
148	Hash	r	Hash value
148	Name	r	Current name of the object

### 7.14.4 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 (149) hash list. It also has no name when created in this way.

**See also:** TFPHashObject.Name (148), TFPHashObject.ChangeOwner (147), TFPHashObject.ChangeOwnerAndName (148)

### 7.14.5 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 (149).

**See also:** CreateNotOwned (147), TFPHashObject.ChangeOwner (147), TFPHashObject.Name (148)

### 7.14.6 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 (148), Name (148)

### 7.14.7 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 ([147](#)), Name ([148](#))

### 7.14.8 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 ([147](#)), ChangeOwnerAndName ([148](#)), Name ([148](#))

### 7.14.9 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 ([148](#)), ChangeOwnerAndName ([148](#))

### 7.14.10 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 ([148](#))

## 7.15 TFPHashObjectList

### 7.15.1 Method overview

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

### 7.15.2 Property overview

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

### 7.15.3 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 (149), TFPHashObjectList.OwnsObjects (155)

### 7.15.4 TFPHashObjectList.Destroy

**Synopsis:** Removes an instance of the hashlist from the heap

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

**Visibility:** `public`

**Description:** `Destroy` cleans up the memory structures maintained by the hashlist and removes the `TFPHashObjectList` instance from the heap. If the list owns its objects, they are freed from memory as well.

`Destroy` should not be called directly, it's better to use `Free` or `FreeAndNil` instead.

**See also:** [TFPHashObjectList.Create \(149\)](#), [TFPHashObjectList.Clear \(150\)](#)

### 7.15.5 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` ([155](#)) is `True`. It always frees all memory needed to contain the objects.

**Errors:** None.

**See also:** [TFPHashObjectList.Extract \(152\)](#), [TFPHashObjectList.Remove \(152\)](#), [TFPHashObjectList.Delete \(151\)](#), [TFPHashObjectList.Add \(150\)](#)

### 7.15.6 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 \(152\)](#), [TFPHashObjectList.Remove \(152\)](#), [TFPHashObjectList.Delete \(151\)](#)

### 7.15.7 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 \(151\)](#), [TFPHashObjectList.Find \(152\)](#), [TFPHashObjectList.FindIndexOf \(153\)](#), [TFPHashObjectList.FindWithHash \(153\)](#)

### 7.15.8 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.HashOfName (149), TFPHashObjectList.Find (152), TFPHashObjectList.FindIndexOf (153), TFPHashObjectList.FindWithHash (153)

### 7.15.9 TFPHashObjectList.GetNextCollision

**Synopsis:** Get next collision number

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

**Visibility:** public

**Description:** Get next collision number

### 7.15.10 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 (155) is True, then the object itself is also freed from memory.

**See also:** TFPHashObjectList.Extract (152), TFPHashObjectList.Remove (152), TFPHashObjectList.Add (150), OwnsObjects (155)

### 7.15.11 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 (150)

### 7.15.12 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` (155).

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

**See also:** [TFPHashObjectList.Delete \(151\)](#), [TFPHashObjectList.Remove \(152\)](#), [TFPHashObjectList.Clear \(150\)](#)

### 7.15.13 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` (155) is `True`, then the object itself is also freed from memory.

**See also:** [TFPHashObjectList.Delete \(151\)](#), [TFPHashObjectList.Clear \(150\)](#), [TFPHashObjectList.Extract \(152\)](#)

### 7.15.14 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 \(151\)](#), [TFPHashObjectList.NameOfIndex \(150\)](#), [TFPHashObjectList.Find \(152\)](#), [TFPHashObjectList.FindIndexOf \(153\)](#), [TFPHashObjectList.FindWithHash \(153\)](#)

### 7.15.15 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 \(151\)](#), [TFPHashObjectList.NameOfIndex \(150\)](#), [TFPHashObjectList.IndexOf \(152\)](#), [TFPHashObjectList.FindIndexOf \(153\)](#), [TFPHashObjectList.FindWithHash \(153\)](#)

### 7.15.16 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 ([151](#)), TFPHashObjectList.NameOfIndex ([150](#)), TFPHashObjectList.IndexOf ([152](#)), TFPHashObjectList.Find ([152](#)), TFPHashObjectList.FindWithHash ([153](#))

### 7.15.17 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 ([151](#)), TFPHashObjectList.NameOfIndex ([150](#)), TFPHashObjectList.IndexOf ([152](#)), TFPHashObjectList.Find ([152](#)), TFPHashObjectList.FindIndexOf ([153](#))

### 7.15.18 TFPHashObjectList.Rename

**Synopsis:** Rename a key

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

**Visibility:** public

**Description:** Rename renames key AOldname to ANewName. The hash value is recomputed and the object is moved in the list to it's new position.

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

### 7.15.19 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.

### 7.15.20 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 ([150](#))

### 7.15.21 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

### 7.15.22 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.

### 7.15.23 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 ([155](#)), Items ([155](#))

### 7.15.24 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 (154), Items (155)

### 7.15.25 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 cannot be changed during the lifetime of the hash list.

**See also:** TFPHashObjectList.Create (149)

### 7.15.26 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 (155).

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

**See also:** Capacity (154), Count (155)

### 7.15.27 TFPHashObjectList.List

**Synopsis:** Low-level hash list

**Declaration:** Property List : TFPHashList

**Visibility:** public

**Access:** Read

**Description:** List exposes the low-level hash list (139). It should not be used directly.

**See also:** TFPHashList (139)

## 7.16 TFPObjectHashTable

### 7.16.1 Description

TFPStringHashTable is a TFPCustomHashTable (132) 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 (165), or for arbitrary pointer data, TFPDataHashTable (138) is more suitable. The objects are exposed with their keys through the Items (157) property.

See also: TFPStringHashTable (165), TFPDataHashTable (138), TFPObjectHashTable.Items (157)

### 7.16.2 Method overview

Page	Property	Description
157	Add	Add a new object to the hash table
156	Create	Create a new instance of TFPObjectHashTable
156	CreateWith	Create a new hash table with given size and hash function
157	Iterate	Iterate over the objects in the hash table

### 7.16.3 Property overview

Page	Property	Access	Description
157	Items	rw	Key-based access to the objects
158	OwnsObjects	rw	Does the hash table own the objects ?

### 7.16.4 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 (158) 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 (158), TFPObjectHashTable.CreateWith (156), TFPObjectHashTable.Items (157)

### 7.16.5 TFPObjectHashTable.CreateWith

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

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

Visibility: public

Description: CreateWith sets the OwnsObjects (158) 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](#) (158), [TFPObjectHashTable.Create](#) (156), [TFPObjectHashTable.Items](#) (157)

### 7.16.6 TFPObjectHashTable.Iterate

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

**Declaration:** function `Iterate(aMethod: TObjectIteratorMethod) : 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.

**See also:** [ForeachCall](#) (119)

### 7.16.7 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](#) (157)

### 7.16.8 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](#) (157)

### 7.16.9 **TFPObjectHashTable.OwnsObjects**

**Synopsis:** Does the hash table own the objects ?

**Declaration:** Property OwnsObjects : Boolean

**Visibility:** public

**Access:** Read,Write

**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 \(156\)](#), [TFPObjectHashTable.Items \(157\)](#)

## 7.17 **TFPObjectList**

### 7.17.1 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 (171), 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 \(171\)](#)

### 7.17.2 Method overview

Page	Property	Description
<a href="#">159</a>	Add	Add an object to the list.
<a href="#">163</a>	Assign	Copy the contents of a list.
<a href="#">159</a>	Clear	Clear all elements in the list.
<a href="#">159</a>	Create	Create a new object list
<a href="#">160</a>	Delete	Delete an element from the list.
<a href="#">159</a>	Destroy	Clears the list and destroys the list instance
<a href="#">160</a>	Exchange	Exchange the location of two objects
<a href="#">160</a>	Expand	Expand the capacity of the list.
<a href="#">161</a>	Extract	Extract an object from the list
<a href="#">161</a>	FindInstanceOf	Search for an instance of a certain class
<a href="#">162</a>	First	Return the first non-nil object in the list
<a href="#">164</a>	ForEachCall	For each object in the list, call a method or procedure, passing it the object.
<a href="#">161</a>	IndexOf	Search for an object in the list
<a href="#">162</a>	Insert	Insert a new object in the list
<a href="#">162</a>	Last	Return the last non-nil object in the list.
<a href="#">163</a>	Move	Move an object to another location in the list.
<a href="#">163</a>	Pack	Remove all Nil references from the list
<a href="#">161</a>	Remove	Remove an item from the list.
<a href="#">163</a>	Sort	Sort the list of objects

### 7.17.3 Property overview

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

### 7.17.4 TFPOBJECTLIST.Create

Synopsis: Create a new object list

Declaration: constructor Create  
constructor Create(FreeObjects: Boolean)

Visibility: public

Description: Create instantiates a new object list. The FreeObjects parameter determines whether objects that are removed from the list should also be freed from memory. By default this is True. This behaviour can be changed after the list was instantiated.

Errors: None.

See also: TFPOBJECTLIST.Destroy (159), TFPOBJECTLIST.OwnsObjects (165), TObjectList (171)

### 7.17.5 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 (165) is True.

See also: TFPOBJECTLIST.OwnsObjects (165), TObjectList.Create (172)

### 7.17.6 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 (165) is True.

See also: TObjectList.Destroy (171)

### 7.17.7 TFPOBJECTLIST.Add

Synopsis: Add an object to the list.

Declaration: function Add(AObject: TObject) : Integer

Visibility: public

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

Note that when OwnsObjects (165) 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 (165), TFPObjectList.Delete (160)

### 7.17.8 **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 (165) is True, the object is also freed.

**Errors:** An access violation may occur when OwnsObjects (165) is True and either the object was freed externally, or when the same object is in the same list twice.

**See also:** TFPObjectList.Remove (161), TFPObjectList.Extract (161), TFPOBJECTLIST.OwnsObjects (165), TFPOBJECTLIST.Add (159), TFPOBJECTLIST.Clear (159)

### 7.17.9 **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 (159), TFPOBJECTLIST.Delete (160)

### 7.17.10 **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 (163), TFPOBJECTLIST.Clear (159), #rtl.classes.tfplist.expand (??)

### 7.17.11 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 \(163\)](#), [TFPObjectList.Clear \(159\)](#), [TFPObjectList.Remove \(161\)](#), [TFPObjectList.Delete \(160\)](#)

### 7.17.12 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 (165) 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 \(163\)](#), [TFPObjectList.Clear \(159\)](#), [TFPObjectList.Delete \(160\)](#), [TFPObjectList.Extract \(161\)](#)

### 7.17.13 TFPObjectList.IndexOf

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

**Declaration:** function IndexOf (AObject: TObject) : Integer

**Visibility:** public

**Description:** IndexOf searches for the presence of AObject in the list, and returns the location (index) in the list. The index is 0-based, and -1 is returned if AObject was not found in the list.

**Errors:** None.

**See also:** [TFPObjectList.Items \(165\)](#), [TFPObjectList.Remove \(161\)](#), [TFPObjectList.Extract \(161\)](#)

### 7.17.14 TFPObjectList.FindInstanceOf

**Synopsis:** Search for an instance of a certain class

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

**Visibility:** public

**Description:** `FindInstanceOf` will look through the instances in the list and will return the 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:** `TFPObjectList.IndexOf` ([161](#))

### **7.17.15 TFPObjectList.Insert**

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

**Declaration:** `procedure Insert (Index: Integer; AObject: TObject)`

**Visibility:** public

**Description:** `Insert` inserts `AObject` at position `Index` in the list. All elements in the list after this position are shifted. The index is zero based, i.e. an insert at position 0 will insert an object at the first position of the list.

**Errors:** None.

**See also:** `TFPObjectList.Add` ([159](#)), `TFPObjectList.Delete` ([160](#))

### **7.17.16 TFPObjectList.First**

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

**Declaration:** `function First : TObject`

**Visibility:** public

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

**Errors:** None.

**See also:** `TFPObjectList.Last` ([162](#)), `TFPObjectList.Pack` ([163](#))

### **7.17.17 TFPObjectList.Last**

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

**Declaration:** `function Last : TObject`

**Visibility:** public

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

**Errors:** None.

**See also:** `TFPObjectList.First` ([162](#)), `TFPObjectList.Pack` ([163](#))

### 7.17.18 **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 (160), 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 (160), TFPObjectList.Delete (160), TFPObjectList.Insert (162)

### 7.17.19 **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.

### 7.17.20 **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 (162), TFPObjectList.Last (162)

### 7.17.21 **TFPObjectList.Sort**

**Synopsis:** Sort the list of objects

**Declaration:** procedure Sort (Compare: TListSortCompare)

**Visibility:** public

**Description:** Sort will perform a quick-sort on the list, using Compare as the compare algorithm. This function should accept 2 pointers and should return the following result:

**less than 0**If the first pointer comes before the second.

**equal to 0**If the pointers have the same value.

**larger than 0**If the first pointer comes after the second.

The function should be able to deal with `Nil` values.

Errors: None.

See also: #rtl.classes.TList.Sort ([??](#))

### 7.17.22 **TFPObjectList.ForEachCall**

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

Declaration: `procedure ForEachCall(proc2call: TObjectListCallback; arg: pointer)`  
`procedure ForEachCall(proc2call: TObjectListStaticCallback; arg: pointer)`

Visibility: public

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

Errors: None.

See also: `TObjectListStaticCallback` ([123](#)), `TObjectListCallback` ([122](#))

### 7.17.23 **TFPObjectList.Capacity**

Synopsis: Capacity of the list

Declaration: Property `Capacity : Integer`

Visibility: public

Access: Read,Write

Description: `Capacity` is the number of elements that the list can contain before it needs to expand itself, i.e., reserve more memory for pointers. It is always equal or larger than `Count` ([164](#)).

See also: `TFPObjectList.Count` ([164](#))

### 7.17.24 **TFPObjectList.Count**

Synopsis: Number of elements in the list.

Declaration: Property `Count : Integer`

Visibility: public

Access: Read,Write

Description: `Count` is the number of elements in the list. Note that this includes `Nil` elements.

See also: `TFPObjectList.Capacity` ([164](#))

### 7.17.25 **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 (159), TFPObjectList.Delete (160), TFPObjectList.Remove (161), TFPObjectList.Clear (159)

### 7.17.26 **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 (164)

### 7.17.27 **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 (??)

## 7.18 **TFPStringHashTable**

### 7.18.1 **Description**

TFPStringHashTable is a TFPCustomHashTable (132) descendent which stores simple strings together with the keys. In case the data associated with the keys are objects, it's better to use TFPOBJObjectHashTable (156), or for arbitrary pointer data, TFPDataHashTable (138) is more suitable. The strings are exposed with their keys through the Items (166) property.

**See also:** TFPOBJObjectHashTable (156), TFPDataHashTable (138), Items (166)

### 7.18.2 Method overview

Page	Property	Description
<a href="#">166</a>	Add	Add a new string to the hash list
<a href="#">166</a>	Iterate	Iterate over the strings in the hash table

### 7.18.3 Property overview

Page	Property	Access	Description
<a href="#">166</a>	Items	rw	Key based access to the strings in the hash table

### 7.18.4 TFPStringHashTable.Iterate

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

**Declaration:** function Iterate(aMethod: TStringIteratorMethod) : 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.

**See also:** ForeachCall ([119](#))

### 7.18.5 TFPStringHashTable.Add

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

**Declaration:** procedure Add(const aKey: string; const aItem: string); Virtual

**Visibility:** public

**Description:** Add adds a new string AItem to the hash list with key AKey.

**Errors:** If a string with key Akey already exists in the hash table, an exception will be raised.

**See also:** TFPStringHashTable.Items ([166](#))

### 7.18.6 TFPStringHashTable.Items

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

**Declaration:** Property Items[index: string]: string; default

**Visibility:** public

**Access:** Read,Write

**Description:** Items provides access to the strings in the hash table using their key: the array index Index is the key. A key which is not present will result in an empty string.

**See also:** TFPStringHashTable.Add ([166](#))

## 7.19 THTCustomNode

### 7.19.1 Description

THTCustomNode is used by the TFPCustomHashTable (132) class to store the keys and associated values.

See also: TFPCustomHashTable (132)

### 7.19.2 Method overview

Page	Property	Description
167	CreateWith	Create a new instance of THTCustomNode
167	HasKey	Check whether this node matches the given key.

### 7.19.3 Property overview

Page	Property	Access	Description
168	Key	r	Key value associated with this hash item.

### 7.19.4 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 (132) class when needed.

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

See also: TFPCustomHashTable (132)

### 7.19.5 THTCustomNode.HasKey

Synopsis: Check whether this node matches the given key.

Declaration: function HasKey(const AKey: string) : Boolean

Visibility: public

Description: HasKey checks whether this node matches the given key AKey, by comparing it with the stored key. It returns True if it does, False if not.

Errors: None.

See also: THTCustomNode.Key (168)

## 7.19.6 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 ([167](#))

## 7.20 THTDataNode

### 7.20.1 Description

THTDataNode is used by TFPDataHashTable ([138](#)) 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 ([138](#)), THTObjectNode ([168](#)), THTStringNode ([169](#))

### 7.20.2 Property overview

Page	Property	Access	Description
<a href="#">168</a>	Data	rw	Data pointer

### 7.20.3 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.

## 7.21 THTObjectNode

### 7.21.1 Description

THTObjectNode is a THTCustomNode ([167](#)) descendent which holds the data in the TFPObjec-tHashTable ([156](#)) hash table. It exposes a data string.

It should not be necessary to use THTObjectNode directly, it's only for inner use by TFPObjec-tHashTable

See also: TFPObjec-tHashTable ([156](#))

## 7.21.2 Property overview

Page	Property	Access	Description
<a href="#">169</a>	Data	rw	Object instance

## 7.21.3 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 ([157](#))

See also: TFPObjectHashTable ([156](#)), TFPObjectHashTable.Items ([157](#)), THTOwnedObjectNode ([169](#))

## 7.22 THTOwnedObjectNode

### 7.22.1 Description

THTOwnedObjectNode is used instead of THTObjectNode ([168](#)) in case TFPObjectHashTable ([156](#)) owns it's objects. When this object is destroyed, the associated data object is also destroyed.

See also: TFPObjectHashTable ([156](#)), THTObjectNode ([168](#)), TFPObjectHashTable.OwnsObjects ([158](#))

### 7.22.2 Method overview

Page	Property	Description
<a href="#">169</a>	Destroy	Destroys the node and the object.

## 7.22.3 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 ([169](#)), TFPObjectHashTable.OwnsObjects ([158](#))

## 7.23 THTStringNode

### 7.23.1 Description

THTStringNode is a THTCustomNode ([167](#)) descendent which holds the data in the TFPString-HashTable ([165](#)) 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 ([165](#))

### 7.23.2 Property overview

Page	Property	Access	Description
<a href="#">170</a>	Data	rw	String data

### 7.23.3 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 ([166](#))

See also: TFPStringHashTable ([165](#))

## 7.24 TObjectBucketList

### 7.24.1 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 ([170](#))

### 7.24.2 Method overview

Page	Property	Description
<a href="#">170</a>	Add	Add an object to the list
<a href="#">171</a>	Remove	Remove an object from the list

### 7.24.3 Property overview

Page	Property	Access	Description
<a href="#">171</a>	Data	rw	Associative array of data items

### 7.24.4 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 ([171](#)), TObjectBucketList.Remove ([171](#))

### 7.24.5 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 (170), TObjectBucketList.Data (171)

### 7.24.6 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 (170)

## 7.25 TObjectList

### 7.25.1 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 (158), 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 (158), TComponentList (127), TClassList (124)

### 7.25.2 Method overview

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

### 7.25.3 Property overview

Page	Property	Access	Description
<a href="#">175</a>	Items	rw	Indexed access to the elements of the list.
<a href="#">174</a>	OwnsObjects	rw	Should the list free elements when they are removed.

### 7.25.4 TObjectList.create

Synopsis: Create a new object list.

Declaration: `constructor create`  
`constructor create(freeobjects: Boolean)`

Visibility: public

Description: `Create` instantiates a new object list. The `FreeObjects` parameter determines whether objects that are removed from the list should also be freed from memory. By default this is `True`. This behaviour can be changed after the list was instantiated.

Errors: None.

See also: `TObjectList.Destroy` ([171](#)), `TObjectList.OwnsObjects` ([174](#)), `TFPOBJECTLIST` ([158](#))

### 7.25.5 TObjectList.Add

Synopsis: Add an object to the list.

Declaration: `function Add(AObject: TObject) : Integer`

Visibility: public

Description: `Add` overrides the `TList` (??) implementation to accept objects (`AObject`) instead of pointers.

The function returns the index of the position where the object was added.

Errors: If the list must be expanded, and not enough memory is available, an exception may be raised.

See also: `TObjectList.Insert` ([174](#)), `#rtl.classes.TList.Delete` (??), `TObjectList.Extract` ([172](#)), `TObjectList.Remove` ([173](#))

### 7.25.6 TObjectList.Extract

Synopsis: Extract an object from the list.

Declaration: `function Extract(Item: TObject) : TObject`

Visibility: public

Description: `Extract` removes the object `Item` from the list if it is present in the list. Contrary to `Remove` ([173](#)), `Extract` does not free the extracted element if `OwnsObjects` ([174](#)) 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` ([173](#))

### 7.25.7 **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 (174) 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 (172)

### 7.25.8 **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 (173)

### 7.25.9 **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 (173)

### 7.25.10 **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 ([172](#)), TObjectList.Remove ([173](#))

### 7.25.11 **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 ([174](#)), TObjectList.Pack ([171](#))

### 7.25.12 **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 ([174](#)), TObjectList.Pack ([171](#))

### 7.25.13 **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](#) (172), [TObjectList.Delete](#) (171), [TObjectList.Remove](#) (173), [TObjectList.Clear](#) (171)

### 7.25.14 TObjectList.Items

**Synopsis:** Indexed access to the elements of the list.

**Declaration:** `Property Items [Index: Integer]: TObject; default`

**Visibility:** public

**Access:** Read,Write

**Description:** `Items` is the default property of the list. It provides indexed access to the elements in the list. The index `Index` is zero based, i.e., runs from 0 (zero) to `Count - 1`.

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

## 7.26 TObjectQueue

### 7.26.1 Method overview

Page	Property	Description
<a href="#">176</a>	Peek	Look at the first object in the queue.
<a href="#">175</a>	Pop	Pop the first element off the queue
<a href="#">175</a>	Push	Push an object on the queue

### 7.26.2 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](#) (175), [TObjectQueue.Peek](#) (176)

### 7.26.3 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](#) (175), [TObjectQueue.Peek](#) (176)

## 7.26.4 TObjectQueue.Peek

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

**Declaration:** function Peek : TObject

**Visibility:** public

**Description:** Peek returns the first object in the queue, without removing it from the queue. If there are no more objects in the queue, Nil is returned.

**Errors:** None

**See also:** TObjectQueue.Push ([175](#)), TObjectQueue.Pop ([175](#))

## 7.27 TObjectStack

### 7.27.1 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 ([177](#)), TStack ([179](#)), TQueue ([179](#)), TObjectQueue ([175](#))

### 7.27.2 Method overview

Page	Property	Description
<a href="#">177</a>	Peek	Look at the top object in the stack.
<a href="#">176</a>	Pop	Pop the top object of the stack.
<a href="#">176</a>	Push	Push an object on the stack.

### 7.27.3 TObjectStack.Push

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

**Declaration:** function Push(AObject: TObject) : TObject

**Visibility:** public

**Description:** Push pushes another object on the stack. It overrides the Push method as implemented in TStack so it accepts only objects as arguments.

**Errors:** If not enough memory is available to expand the stack, an exception may be raised.

**See also:** TObjectStack.Pop ([176](#)), TObjectStack.Peek ([177](#))

### 7.27.4 TObjectStack.Pop

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

**Declaration:** function Pop : TObject

**Visibility:** public

**Description:** `Pop` pops the top object of the stack, and returns the object instance. If there are no more objects on the stack, `Nil` is returned.

**Errors:** None

**See also:** `TObjectStack.Push` (176), `TObjectStack.Peek` (177)

### 7.27.5 `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` (176), `TObjectStack.Pop` (176)

## 7.28 `TOrderedList`

### 7.28.1 Description

`TOrderedList` provides the base class for `TQueue` (179) and `TStack` (179). 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` (179), `TStack` (179)

### 7.28.2 Method overview

Page	Property	Description
178	<code>AtLeast</code>	Check whether the list contains a certain number of elements.
178	<code>Count</code>	Number of elements on the list.
177	<code>Create</code>	Create a new ordered list
178	<code>Destroy</code>	Free an ordered list
179	<code>Peek</code>	Return the next element to be popped from the list.
179	<code>Pop</code>	Remove an element from the list.
178	<code>Push</code>	Push another element on the list.

### 7.28.3 `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` (178)

#### 7.28.4 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` (177)

#### 7.28.5 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` (178)

#### 7.28.6 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` (178)

#### 7.28.7 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` (179), `TOrderedList.Peek` (179)

## 7.28.8 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 ([179](#)), TOrderedList.Push ([178](#))

## 7.28.9 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 ([179](#)), without actually popping it from the list.

**Errors:** None.

See also: TOrderedList.Pop ([179](#)), TOrderedList.Push ([178](#))

## 7.29 TQueue

### 7.29.1 Description

TQueue is a descendent of TOrderedList ([177](#)) which implements Push ([178](#)) and Pop ([179](#)) behaviour as a queue: what is first pushed on the queue, is popped off first (FIFO: First in, first out).

TQueue offers no new methods, it merely implements some abstract methods introduced by TOrderedList ([177](#))

See also: TOrderedList ([177](#)), TObjectQueue ([175](#)), TStack ([179](#))

## 7.30 TStack

### 7.30.1 Description

TStack is a descendent of TOrderedList ([177](#)) which implements Push ([178](#)) and Pop ([179](#)) behaviour as a stack: what is last pushed on the stack, is popped off first (LIFO: Last in, first out).

TStack offers no new methods, it merely implements some abstract methods introduced by TOrderedList ([177](#))

See also: TOrderedList ([177](#)), TObjectStack ([176](#)), TQueue ([179](#))

# Chapter 8

## Reference for unit 'CustApp'

### 8.1 Used units

Table 8.1: Used units by unit 'CustApp'

Name	Page
Classes	??
System	??
sysutils	??

### 8.2 Overview

The `CustApp` unit implements the `TCustomApplication` (181) class, which serves as the common ancestor to many kinds of `TApplication` classes: a GUI application in the LCL, a CGI application in FPCGI, a daemon application in daemonapp. It introduces some properties to describe the environment in which the application is running (environment variables, program command-line parameters) and introduces some methods to initialize and run a program, as well as functionality to handle exceptions.

Typical use of a descendent class is to introduce a global variable `Application` and use the following code:

```
Application.Initialize;  
Application.Run;
```

Since normally only a single instance of this class is created, and it is a `TComponent` descendent, it can be used as an owner for many components, doing so will ensure these components will be freed when the application terminates.

### 8.3 Constants, types and variables

#### 8.3.1 Types

```
TEventLogTypes = Set of TEventType
```

TEventLogTypes is a set of TEventType (??), used in TCustomApplication.EventLogFilter (190) 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.

### 8.3.2 Variables

```
CustomApplication : TCustomApplication = Nil
```

CustomApplication contains the global application instance. All descendants of TCustomApplication (181) 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.

## 8.4 TCustomApplication

### 8.4.1 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 (182), ShowException (183), OnException (188) and StopOnException (190).
- Command-line parameter parsing in FindOptionIndex (184), GetOptionValue (184), CheckOptions (185) and HasOption (185)
- Environment variable handling in GetEnvironmentList (186) and EnvironmentVariable (189).

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

### 8.4.2 Method overview

Page	Property	Description
185	CheckOptions	Check whether all given options on the command-line are valid.
182	Create	Create a new instance of the TCustomApplication class
182	Destroy	Destroys the TCustomApplication instance.
184	FindOptionIndex	Return the index of an option.
186	GetEnvironmentList	Return a list of environment variables.
184	GetOptionValue	Return the value of a command-line option.
182	HandleException	Handle an exception.
185	HasOption	Check whether an option was specified.
183	Initialize	Initialize the application
186	Log	Write a message to the event log
183	Run	Runs the application.
183	ShowException	Show an exception to the user
184	Terminate	Terminate the application.

### 8.4.3 Property overview

Page	Property	Access	Description
190	CaseSensitiveOptions	rw	Are options interpreted case sensitive or not
188	ConsoleApplication	r	Is the application a console application or not
189	EnvironmentVariable	r	Environment variable access
190	EventLogFilter	rw	Event to filter events, before they are sent to the system log
187	ExeName	r	Name of the executable.
187	HelpFile	rw	Location of the application help file.
188	Location	r	Application location
188	OnException	rw	Exception handling event
189	OptionChar	rw	Command-line switch character
189	ParamCount	r	Number of command-line parameters
189	Params	r	Command-line parameters
190	StopOnException	rw	Should the program loop stop on an exception
187	Terminated	r	Was Terminate called or not
187	Title	rw	Application title

### 8.4.4 TCustomApplication.Create

**Synopsis:** Create a new instance of the TCustomApplication class

**Declaration:** constructor Create(AOwner: TComponent); Override

**Visibility:** public

**Description:** Create creates a new instance of the TCustomApplication class. It sets some defaults for the various properties, and then calls the inherited Create.

**See also:** TCustomApplication.Destroy (182)

### 8.4.5 TCustomApplication.Destroy

**Synopsis:** Destroys the TCustomApplication instance.

**Declaration:** destructor Destroy; Override

**Visibility:** public

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

**See also:** TCustomApplication.Create (182)

### 8.4.6 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 (188) 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` (183) 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` (190) property is set to `True`, the `Terminated` (187) property is set to `True`, which will cause the `Run` (183) loop to stop, and the application will exit.

See also: `ShowException` (183), `StopOnException` (190), `Terminated` (187), `Run` (183)

#### 8.4.7 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` (183), `TCustomApplication.Terminated` (187)

#### 8.4.8 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` (182). If `TCustomApplication.StopOnException` (190) 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` (182), `TCustomApplication.StopException` (181)

#### 8.4.9 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 (182), TCustomApplication.StopException (181)

#### 8.4.10 TCustomApplication.Terminate

Synopsis: Terminate the application.

Declaration: procedure Terminate; Virtual

Visibility: public

Description: Terminate sets the Terminated property to True. By itself, this does not terminate the application. Instead, descendant classes should in their DoRun method, check the value of the Terminated (187) property and properly shut down the application if it is set to True.

See also: TCustomApplication.Terminated (187), TCustomApplication.Run (183)

#### 8.4.11 TCustomApplication.FindOptionIndex

Synopsis: Return the index of an option.

Declaration: function FindOptionIndex(const S: string; var Longopt: Boolean) : 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 (190) property, the search is performed case sensitive or case insensitive.

Options are identified as command-line parameters which start with OptionChar (189) (by default the dash ('-') character).

See also: HasOption (185), GetOptionValue (184), CheckOptions (185), CaseSensitiveOptions (190), OptionChar (189)

#### 8.4.12 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 (190) property, the search is performed case sensitive or case insensitive.

Options are identified as command-line parameters which start with OptionChar (189) (by default the dash ('-') character).

**See also:** FindOptionIndex (184), HasOption (185), CheckOptions (185), CaseSensitiveOptions (190), OptionChar (189)

#### 8.4.13 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 (190) property, the search is performed case sensitive or case insensitive.

Options are identified as command-line parameters which start with OptionChar (189) (by default the dash ('-') character).

**See also:** FindOptionIndex (184), GetOptionValue (184), CheckOptions (185), CaseSensitiveOptions (190), OptionChar (189)

#### 8.4.14 TCustomApplication.CheckOptions

**Synopsis:** Check whether all given options on the command-line are valid.

**Declaration:** `function CheckOptions(const ShortOptions: string;  
const Longopts: TStrings; Opts: TStrings;  
NonOpts: TStrings; AllErrors: Boolean) : string  
function CheckOptions(const ShortOptions: string;  
const Longopts: Array of string; Opts: TStrings;  
NonOpts: TStrings; AllErrors: Boolean) : string  
function CheckOptions(const ShortOptions: string;  
const Longopts: TStrings; AllErrors: Boolean)  
: string  
function CheckOptions(const ShortOptions: string;  
const LongOpts: Array of string; AllErrors: Boolean)  
: string  
function CheckOptions(const ShortOptions: string; const LongOpts: string;  
AllErrors: Boolean) : string`

**Visibility:** public

**Description:** CheckOptions scans the command-line and checks whether the options given are valid options. It also checks whether options that require a valued are indeed specified with a value.

The `ShortOptions` contains a string with valid short option characters. Each character in the string is a valid option character. If a character is followed by a colon (:), then a value must be specified. If it is followed by 2 colon characters (::) then the value is optional.

`LongOpts` is a list of strings (which can be specified as an array, a `TStrings` instance or a string with whitespace-separated values) of valid long options.

When the function returns, if `Opts` is non-`Nil`, the `Opts` stringlist is filled with the passed valid options. If `NonOpts` is non-`nil`, it is filled with any non-option strings that were passed on the command-line.

The function returns an empty string if all specified options were valid options, and whether options requiring a value have a value. If an error was found during the check, the return value is a string describing the error.

Options are identified as command-line parameters which start with `OptionChar` (189) (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` (184), `GetOptionValue` (184), `HasOption` (185), `CaseSensitiveOptions` (190), `OptionChar` (189)

#### 8.4.15 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` (189)

#### 8.4.16 TCustomApplication.Log

**Synopsis:** Write a message to the event log

**Declaration:** `procedure Log(EventType: TEventType; const Msg: string)`

**Visibility:** public

**Description:** `Log` is meant for all applications to have a default logging mechanism. By default it does not do anything, descendant 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` (432)

**Errors:** None.

**See also:** `#rtl.sysutils.TEventType` (??)

#### **8.4.17 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 (??)

#### **8.4.18 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 ([187](#))

#### **8.4.19 TCustomApplication.Terminated**

**Synopsis:** Was Terminate called or not

**Declaration:** Property Terminated : Boolean

**Visibility:** public

**Access:** Read

**Description:** Terminated indicates whether Terminate ([184](#)) 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 ([184](#))

#### **8.4.20 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 (187)

#### 8.4.21 TCustomApplication.OnException

Synopsis: Exception handling event

Declaration: Property OnException : TExceptionEvent

Visibility: public

Access: Read,Write

Description: OnException can be set to provide custom handling of events, instead of the default action, which is simply to show the event using ShowEvent (181).

If set, OnException is called by the HandleEvent (181) routine. Do not use the OnException event directly, instead call HandleEvent

See also: ShowEvent (181)

#### 8.4.22 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 (??)

#### 8.4.23 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 under certain circumstances). See the discussion of Paramstr (??) in the RTL documentation.

See also: Paramstr (??), Params (189)

#### 8.4.24 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 (189). It is equivalent to calling ParamStr (??).

See also: ParamCount (189), Paramstr (??)

#### 8.4.25 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 (189) property.

See also: Params (189), Paramstr (??), ParamCount (??)

#### 8.4.26 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 (186) can be used.

See also: GetEnvironmentList (186), TCustomApplication.Params (189)

#### 8.4.27 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-alphanumeric character (although no check is performed on this).

See also: [FindOptionIndex \(184\)](#), [GetOptionValue \(184\)](#), [HasOption \(185\)](#), [CaseSensitiveOptions \(190\)](#), [CheckOptions \(185\)](#)

#### 8.4.28 TCustomApplication.CaseSensitiveOptions

Synopsis: Are options interpreted case sensitive or not

Declaration: Property CaseSensitiveOptions : Boolean

Visibility: public

Access: Read,Write

Description: CaseSensitiveOptions determines whether FindOptionIndex (184) and CheckOptions (185) 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 \(184\)](#), [GetOptionValue \(184\)](#), [HasOption \(185\)](#), [OptionChar \(189\)](#), [CheckOptions \(185\)](#)

#### 8.4.29 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 (183) and HandleException (182) procedures in case of an unhandled exception in the DoRun code. If StopOnException is True then Terminate (184) will be called after the exception was handled.

See also: [Run \(183\)](#), [HandleException \(182\)](#), [Terminate \(184\)](#)

#### 8.4.30 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 (186) 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 \(186\)](#)

# Chapter 9

## Reference for unit 'daemonapp'

### 9.1 Used units

Table 9.1: Used units by unit 'daemonapp'

Name	Page
Classes	??
CustApp	<a href="#">180</a>
eventlog	<a href="#">430</a>
rtlconsts	??
System	??
sysutils	??

### 9.2 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 descendants of this class can be implemented and instantiated in a single daemon application. For more information, see `TDaemon` ([207](#)).

**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` ([212](#)).

**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` ([215](#)).

As can be seen, a single application can implement one or 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` ([432](#)) 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.

## **9.3 Daemon application architecture**

[Still needs to be completed]

## **9.4 Constants, types and variables**

### **9.4.1 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

### 9.4.2 Types

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

Table 9.2: Enumeration values for type TCurrentStatus

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

TCurrentStatus indicates the current state of the daemon. It changes from one state to the next during the time the instance is active. The daemon application changes the state of the daemon, depending on signals it gets from the operating system, by calling the appropriate methods.

---

```
TCustomControlCodeEvent = procedure(Sender: TCustomDaemon; ACode: DWord;
                                     var Handled: Boolean) of object
```

In case the system sends a non-standard control code to the daemon, an event handler is executed with this prototype.

```
TCustomDaemonApplicationClass = Class of TCustomDaemonApplication
```

Class pointer for TCustomDaemonApplication

```
TCustomDaemonClass = Class of TCustomDaemon
```

The class type is needed in the TDaemonDef ([215](#)) definition.

```
TCustomDaemonMapperClass = Class of TCustomDaemonMapper
```

TCustomDaemonMapperClass is the class of TCustomDaemonMapper. It is used in the RegisterDaemonMapper ([197](#)) 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 ([198](#)) 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 9.3: Enumeration values for type TDaemonOption

Value	Explanation
doAllowPause	The daemon can be paused.
doAllowStop	The daemon can be stopped.
doInteractive	The daemon interacts with the desktop.

Enumerated that enumerates the various daemon operation options.

```
TDaemonOptions = Set of TDaemonOption
```

TDaemonOption enumerates the various options a daemon can have.

```
TDaemonRunMode = (drmUnknown, drmInstall, drmUninstall, drmRun)
```

Table 9.4: Enumeration values for type TDaemonRunMode

Value	Explanation
drmInstall	Daemon install mode (windows only)
drmRun	Daemon is running normally
drmUninstall	Daemon uninstall mode (windows only)
drmUnknown	Unknown mode

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

```
TErrorSeverity = (esIgnore, esNormal, esSevere, esCritical)
```

Table 9.5: Enumeration values for type TErrorSeverity

Value	Explanation
esCritical	Error is logged, and startup is stopped if last known good configuration is active, or system is restarted using last known good configuration
esIgnore	Ignore startup errors
esNormal	Error is logged, but startup continues
esSevere	Error is logged, and startup is continued if last known good configuration is active, or system is restarted using last known good configuration

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 9.6: Enumeration values for type TServiceType

Value	Explanation
stDevice	Device driver
stFileSystem	File system driver
stWin32	Regular win32 service

The type of service. This type is used on windows only, to signal the operating system what kind of service is being installed or run.

```
TStartType = (stBoot, stSystem, stAuto, stManual, stDisabled)
```

Table 9.7: Enumeration values for type TStartType

Value	Explanation
stAuto	Started automatically by service manager during system startup
stBoot	During system boot
stDisabled	Service is not started, it is disabled
stManual	Started manually by the user or other processes.
stSystem	During load of device drivers

TStartType can be used to define when the service must be started on windows. This type is not used on other platforms.

### 9.4.3 Variables

AppClass : TCustomDaemonApplicationClass

AppClass can be set to the class of a TCustomDaemonApplication (200) descendant. When the Application (196) 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',

Names for various service statuses

DefaultDaemonOptions : TDaemonOptions = [doAllowStop, doAllowPause]

DefaultDaemonOptions are the default options with which a daemon definition (TDaemonDef (215)) is created.

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

Status message

## 9.5 Procedures and functions

### 9.5.1 Application

Synopsis: Application instance

Declaration: function Application : TCustomDaemonApplication

Visibility: default

Description: Application is the TCustomDaemonApplication (200) instance used by this application. The instance is created at the first invocation of this function, so it is possible to use RegisterDaemonApplicationClass (197) to register an alternative TCustomDaemonApplication class to run the application.

See also: TCustomDaemonApplication (200), RegisterDaemonApplicationClass (197)

### **9.5.2 DaemonError**

**Synopsis:** Raise an EDaemon exception

**Declaration:** `procedure DaemonError(Msg: string)`  
`procedure DaemonErrorFmt(Fmt: string; Args: Array of const)`

**Visibility:** default

**Description:** `DaemonError` raises an EDaemon (198) exception with message `Msg` or it formats the message using `Fmt` and `Args`.

**See also:** `EDaemon` (198)

### **9.5.3 RegisterDaemonApplicationClass**

**Synopsis:** Register alternative `TCustomDaemonApplication` class.

**Declaration:** `procedure RegisterDaemonApplicationClass(AClass: TCustomDaemonApplicationClass)`

**Visibility:** default

**Description:** `RegisterDaemonApplicationClass` can be used to register an alternative `TCustomDaemonApplication` (200) descendent which will be used when creating the global Application (196) instance. Only the last registered class pointer will be used.

**See also:** `TCustomDaemonApplication` (200), `Application` (196)

### **9.5.4 RegisterDaemonClass**

**Synopsis:** Register daemon

**Declaration:** `procedure RegisterDaemonClass(AClass: TCustomDaemonClass)`

**Visibility:** default

**Description:** `RegisterDaemonClass` must be called for each `TCustomDaemon` (198) descendent that is used in the class: the class pointer and class name are used by the `TCustomDaemonMapperClass` (194) class to create a `TCustomDaemon` instance when a daemon is required.

**See also:** `TCustomDaemonMapperClass` (194), `TCustomDaemon` (198)

### **9.5.5 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` (200) code, so registering an alternative mapping class should happen in the initialization section of the application units.

**See also:** `TCustomDaemonApplication` (200), `TCustomDaemonMapperClass` (194)

## 9.6 EDaemon

### 9.6.1 Description

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

See also: [DaemonError \(197\)](#)

## 9.7 TCustomDaemon

### 9.7.1 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 ([207](#)) can be designed to implement the behaviour of the daemon.

See also: [TDaemon \(207\)](#), [TDaemonDef \(215\)](#), [TDaemonController \(212\)](#), [TDaemonApplication \(212\)](#)

### 9.7.2 Method overview

Page	Property	Description
<a href="#">198</a>	LogMessage	Log a message to the system log
<a href="#">199</a>	ReportStatus	Report the current status to the operating system

### 9.7.3 Property overview

Page	Property	Access	Description
<a href="#">200</a>	Controller	r	TDaemonController instance controlling this daemon instance
<a href="#">199</a>	DaemonThread	r	Thread in which daemon is running
<a href="#">199</a>	Definition	r	The definition used to instantiate this daemon instance
<a href="#">200</a>	Logger	r	TEventLog instance used to send messages to the system log
<a href="#">200</a>	Status	rw	Current status of the daemon

### 9.7.4 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 ([432](#)) instance is used to actually send messages to the system log.

The message is sent with an 'error' flag (using TEventLog.Error ([435](#))).

Errors: None.

See also: [ReportStatus \(199\)](#)

### 9.7.5 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 ([198](#))

### 9.7.6 TCustomDaemon.Definition

**Synopsis:** The definition used to instantiate this daemon instance

**Declaration:** Property Definition : TDaemonDef

**Visibility:** public

**Access:** Read

**Description:** Definition is the TDaemonDef ([215](#)) 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 ([215](#))

### 9.7.7 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 its 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 ([200](#))

### 9.7.8 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 (199)

### 9.7.9 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 (199)

### 9.7.10 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 (432) instance used to send messages to the system log. It is used by the LogMessage (198) call, but is accessible through the Logger property in case more configurable logging is needed than offered by LogMessage.

See also: LogMessage (198), TEventLog (432)

## 9.8 TCustomDaemonApplication

### 9.8.1 Description

TCustomDaemonApplication is a TCustomApplication (181) 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 (207) instances by checking the TCustomDaemonMapperClass (194) instance that contains the daemon maps.

See also: TCustomApplication (181), TCustomDaemonMapperClass (194)

### 9.8.2 Method overview

Page	Property	Description
<a href="#">201</a>	Create	
<a href="#">202</a>	CreateDaemon	Create daemon instance
<a href="#">203</a>	CreateForm	Create a component
<a href="#">201</a>	Destroy	Clean up the TCustomDaemonApplication instance
<a href="#">202</a>	InstallDaemons	Install all daemons.
<a href="#">202</a>	RunDaemons	Run all daemons.
<a href="#">201</a>	ShowException	Show an exception
<a href="#">203</a>	ShowHelp	Display a help message
<a href="#">202</a>	StopDaemons	Stop all daemons
<a href="#">203</a>	UnInstallDaemons	Uninstall all daemons

### 9.8.3 Property overview

Page	Property	Access	Description
<a href="#">205</a>	AutoRegisterMessageFile	rw	
<a href="#">204</a>	EventLog	r	Event logger instance
<a href="#">204</a>	GuiHandle	rw	Handle of GUI loop main application window handle
<a href="#">204</a>	GUIMainLoop	rw	GUI main loop callback
<a href="#">203</a>	OnRun	rw	Event executed when the daemon is run.
<a href="#">204</a>	RunMode	r	Application mode

### 9.8.4 TCustomDaemonApplication.Create

Declaration: constructor Create(AOwner: TComponent); Override

Visibility: public

### 9.8.5 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 ([204](#))

### 9.8.6 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.

### **9.8.7 TCustomDaemonApplication.CreateDaemon**

**Synopsis:** Create daemon instance

**Declaration:** function CreateDaemon(DaemonDef: TDaemonDef) : TCustomDaemon

**Visibility:** public

**Description:** CreateDaemon is called whenever a TCustomDaemon ([198](#)) instance must be created from a TDaemonDef ([215](#)) daemon definition, passed in DaemonDef. It initializes the TCustomDaemon instance, and creates a controller instance of type TDaemonController ([212](#)) to control the daemon. Finally, it assigns the created daemon to the TDaemonDef.Instance ([216](#)) property.

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

**See also:** TDaemonController ([212](#)), TCustomDaemon ([198](#)), TDaemonDef ([215](#)), TDaemonDef.Instance ([216](#))

### **9.8.8 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 ([213](#)), TCustomDaemonApplication.UnInstallDaemons ([203](#)), TCustomDaemonApplication.RunDaemons ([202](#))

### **9.8.9 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 ([203](#)), TCustomDaemonApplication.RunDaemons ([202](#)), TCustomDaemonApplication.StopDaemons ([202](#))

### **9.8.10 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 ([203](#)), TCustomDaemonApplication.InstallDaemons ([202](#)), TCustomDaemonApplication.StopDaemons ([202](#))

### 9.8.11 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 ([202](#)), TCustomDaemonApplication.InstallDaemons ([202](#)), TCustomDaemonApplication.StopDaemons ([202](#))

### 9.8.12 TCustomDaemonApplication.ShowHelp

**Synopsis:** Display a help message

**Declaration:** procedure ShowHelp

**Visibility:** public

**Description:** ShowHelp displays a help message explaining the command-line options on standard output.

### 9.8.13 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 ([202](#))

### 9.8.14 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 ([202](#)), TCustomDaemonApplication.InstallDaemons ([202](#)), TCustomDaemonApplication.UnInstallDaemons ([203](#))

### **9.8.15 TCustomDaemonApplication.EventLog**

**Synopsis:** Event logger instance

**Declaration:** Property EventLog : TEventLog

**Visibility:** public

**Access:** Read

**Description:** EventLog is the TEventLog (432) instance which is used to log events to the system log with the Log (200) method. It is created when the application instance is created, and destroyed when the application is destroyed.

**See also:** TEventLog (432), Log (200)

### **9.8.16 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 (204)

### **9.8.17 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 (204) procedure exists, and the application creates and runs a message loop by itself.

**See also:** GUIMainLoop (204)

### **9.8.18 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 (202), RunDaemons (202) or UnInstallDaemons (203) is called.

**See also:** InstallDaemons (202), RunDaemons (202), UnInstallDaemons (203)

### 9.8.19 TCustomDaemonApplication.AutoRegisterMessageFile

**Declaration:** Property AutoRegisterMessageFile : Boolean

**Visibility:** public

**Access:** Read,Write

## 9.9 TCustomDaemonMapper

### 9.9.1 Description

The `TCustomDaemonMapper` class is responsible for mapping a daemon definition to an actual `TDaemon` instance. It maintains a `TDaemonDefs` (219) 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` (215) definitions and the actual `TDaemon` (207) instances that will be used to run the daemons.

The `TCustomDaemonMapper` class has no support for streaming. The `TDaemonMapper` (221) class has support for streaming (and hence visual designing).

See also: `TDaemon` (207), `TDaemonDef` (215), `TDaemonDefs` (219), `TDaemonMapper` (221)

### 9.9.2 Method overview

Page	Property	Description
205	Create	Create a new instance of <code>TCustomDaemonMapper</code>
206	Destroy	Clean up and destroy a <code>TCustomDaemonMapper</code> instance.

### 9.9.3 Property overview

Page	Property	Access	Description
206	DaemonDefs	rw	Collection of daemons
206	OnCreate	rw	Event called when the daemon mapper is created.
206	OnDestroy	rw	Event called when the daemon mapper is freed.
207	OnInstall	rw	Event called when the daemons are installed
207	OnRun	rw	Event called when the daemons are executed.
207	OnUnInstall	rw	Event called when the daemons are uninstalled

### 9.9.4 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` (219) 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` (219), `TCustomDaemonApplication` (200), `TCustomDaemonMapper.Destroy` (206)

### 9.9.5 TCustomDaemonMapper.Destroy

**Synopsis:** Clean up and destroy a TCustomDaemonMapper instance.

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

**Visibility:** public

**Description:** `Destroy` frees the DaemonDefs ([206](#)) collection and calls the inherited destructor.

**See also:** TDaemonDefs ([219](#)), TCustomDaemonMapper.Create ([205](#))

### 9.9.6 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 ([200](#))

### 9.9.7 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 ([206](#)), OnUnInstall ([207](#)), OnCreate ([206](#)), OnDestroy ([206](#))

### 9.9.8 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 ([206](#)) event.

**See also:** OnCreate ([206](#)), OnInstall ([207](#)), OnUnInstall ([207](#)), OnCreate ([206](#))

### **9.9.9 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 \(207\)](#), [OnUnInstall \(207\)](#), [OnCreate \(206\)](#), [OnDestroy \(206\)](#)

### **9.9.10 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 \(207\)](#), [OnUnInstall \(207\)](#), [OnCreate \(206\)](#), [OnDestroy \(206\)](#)

### **9.9.11 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 \(207\)](#), [OnInstall \(207\)](#), [OnCreate \(206\)](#), [OnDestroy \(206\)](#)

## **9.10 TDaemon**

### **9.10.1 Description**

TDaemon is a TCustomDaemon ([198](#)) 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 ([199](#))), which is not the main program thread.

**See also:** [TCustomDaemon \(198\)](#), [TDaemonController \(212\)](#)

### 9.10.2 Property overview

Page	Property	Access	Description
211	AfterInstall	rw	Called after the daemon was installed
211	AfterUnInstall	rw	Called after the daemon is uninstalled
210	BeforeInstall	rw	Called before the daemon will be installed
211	BeforeUnInstall	rw	Called before the daemon is uninstalled
208	Definition		
209	OnContinue	rw	Daemon continue
211	OnControlCode	rw	Called when a control code is received for the daemon
210	OnExecute	rw	Daemon execute event
209	OnPause	rw	Daemon pause event
210	OnShutDown	rw	Daemon shutdown
208	OnStart	rw	Daemon start event
209	OnStop	rw	Daemon stop event
208	Status		

### 9.10.3 TDaemon.Definition

Declaration: Property Definition :

Visibility: public

Access:

### 9.10.4 TDaemon.Status

Declaration: Property Status :

Visibility: public

Access:

### 9.10.5 TDaemon.OnStart

Synopsis: Daemon start event

Declaration: Property OnStart : TDaemonOKEvent

Visibility: published

Access: Read,Write

Description: OnStart is the event called when the daemon must be started. This event handler should return as quickly as possible. If it must perform lengthy operations, it is best to report the status to the operating system at regular intervals using the ReportStatus (199) 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 (210) event handler, so the event handler can return at once.

See also: TDaemon.OnStop (209), TDaemon.OnExecute (210), TDaemon.OnContinue (209), ReportStatus (199)

### 9.10.6 TDaemon.OnStop

**Synopsis:** Daemon stop event

**Declaration:** Property OnStop : TDaemonOKEEvent

**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 (199) method.

If a thread was started in the OnExecute (210) event, this is the place where the thread should be stopped.

See also: TDaemon.OnStart (208), TDaemon.OnPause (209), ReportStatus (199)

### 9.10.7 TDaemon.OnPause

**Synopsis:** Daemon pause event

**Declaration:** Property OnPause : TDaemonOKEEvent

**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 (199) method.

If a thread was started in the OnExecute (210) event, this is the place where the thread's execution should be suspended.

See also: TDaemon.OnStop (209), TDaemon.OnContinue (209), ReportStatus (199)

### 9.10.8 TDaemon.OnContinue

**Synopsis:** Daemon continue

**Declaration:** Property OnContinue : TDaemonOKEEvent

**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 (199) method.

If a thread was started in the OnExecute (210) event and it was suspended in a OnPause (208) event, this is the place where the thread's execution should be resumed.

See also: TDaemon.OnStart (208), TDaemon.OnPause (209), ReportStatus (199)

### 9.10.9 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 (210), this is the place where the thread's executed should be stopped or the thread freed from memory.

See also: TDaemon.OnStart (208), TDaemon.OnPause (209), ReportStatus (199)

### 9.10.10 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 (208), TDaemon.OnStop (209)

### 9.10.11 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 (211), BeforeUnInstall (211), AfterUnInstall (211)

### **9.10.12 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 (210), BeforeUnInstall (211), AfterUnInstall (211)

### **9.10.13 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 (210), AfterInstall (211), AfterUnInstall (211)

### **9.10.14 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 (210), AfterInstall (211), BeforeUnInstall (211)

### **9.10.15 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 (192)

## 9.11 TDaemonApplication

### 9.11.1 Description

`TDaemonApplication` is the default `TCustomDaemonApplication` (200) descendent that is used to run the daemon application. It is possible to register an alternative `TCustomDaemonApplication` class (using `RegisterDaemonApplicationClass` (197)) to run the application in a different manner.

See also: `TCustomDaemonApplication` (200), `RegisterDaemonApplicationClass` (197)

## 9.12 TDaemonController

### 9.12.1 Description

`TDaemonController` is a class that is used by the `TDaemonApplication` (212) 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` (198), `TDaemonApplication` (212)

### 9.12.2 Method overview

Page	Property	Description
213	Controller	Controller
212	Create	Create a new instance of the <code>TDaemonController</code> class
213	Destroy	Free a <code>TDaemonController</code> instance.
213	Main	Daemon main entry point
214	ReportStatus	Report the status to the operating system.
213	StartService	Start the service

### 9.12.3 Property overview

Page	Property	Access	Description
215	CheckPoint		Send checkpoint signal to the operating system
214	Daemon	r	Daemon instance this controller controls.
214	LastStatus	r	Last reported status
214	Params	r	Parameters passed to the daemon

### 9.12.4 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` (212) instance, and `AOwner` will be set to the global `TDaemonApplication` (212) instance.

See also: `TDaemonApplication` (212), `Destroy` (213)

### 9.12.5 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 \(212\)](#)

### 9.12.6 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 \(213\)](#)

### 9.12.7 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 \(222\)](#)

### 9.12.8 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 \(211\)](#)

### 9.12.9 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:** TDaemon.ReportStatus ([207](#)), TDaemonController.LastStatus ([214](#))

### 9.12.10 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.

### 9.12.11 TDaemonController.Params

**Synopsis:** Parameters passed to the daemon

**Declaration:** Property Params : TString

**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 ([213](#)) method.

### 9.12.12 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 ([214](#))

### 9.12.13 TDaemonController.CheckPoint

**Synopsis:** Send checkpoint signal to the operating system

**Declaration:** Property CheckPoint : DWord

**Visibility:** public

**Access:**

**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 ([214](#))

## 9.13 TDaemonDef

### 9.13.1 Description

TDaemonDef contains the definition of a daemon in the application: The name of the daemon, which TCustomDaemon ([198](#)) descendant 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 ([212](#)), TDaemon ([207](#))

### 9.13.2 Method overview

Page	Property	Description
<a href="#">215</a>	Create	Create a new TDaemonDef instance
<a href="#">216</a>	Destroy	Free a TDaemonDef from memory

### 9.13.3 Property overview

Page	Property	Access	Description
<a href="#">216</a>	DaemonClass	r	TDaemon class to use for this daemon
<a href="#">216</a>	DaemonClassName	rw	Name of the TDaemon class to use for this daemon
<a href="#">217</a>	Description	rw	Description of the daemon
<a href="#">217</a>	DisplayName	rw	Displayed name of the daemon (service)
<a href="#">218</a>	Enabled	rw	Is the daemon enabled or not
<a href="#">216</a>	Instance	rw	Instance of the daemon class
<a href="#">219</a>	LogStatusReport	rw	Log the status report to the system log
<a href="#">217</a>	Name	rw	Name of the daemon (service)
<a href="#">218</a>	OnCreateInstance	rw	Event called when a daemon is instantiated
<a href="#">218</a>	Options	rw	Service options
<a href="#">217</a>	RunArguments	rw	Additional command-line arguments when running daemon.
<a href="#">218</a>	WinBindings	rw	Windows-specific bindings (windows only)

### 9.13.4 TDaemonDef.Create

**Synopsis:** Create a new TDaemonDef instance

**Declaration:** constructor Create (ACollection: TCollection);   Override

**Visibility:** public

**Description:** Create initializes a new TDaemonDef instance. It should not be necessary to instantiate a definition manually, it is handled by the collection.

**See also:** TDaemonDefs ([219](#))

### **9.13.5 TDaemonDef.Destroy**

**Synopsis:** Free a TDaemonDef from memory

**Declaration:** destructor Destroy;   Override

**Visibility:** public

**Description:** Destroy removes the TDaemonDef from memory.

### **9.13.6 TDaemonDef.DaemonClass**

**Synopsis:** TDaemon class to use for this daemon

**Declaration:** Property DaemonClass : TCustomDaemonClass

**Visibility:** public

**Access:** Read

**Description:** DaemonClass is the TDaemon class that is used when this service is requested. It is looked up in the application's global daemon mapper by its name in DaemonClassName ([216](#)).

**See also:** DaemonClassName ([216](#)), TDaemonMapper ([221](#))

### **9.13.7 TDaemonDef.Instance**

**Synopsis:** Instance of the daemon class

**Declaration:** Property Instance : TCustomDaemon

**Visibility:** public

**Access:** Read,Write

**Description:** Instance points to the TDaemon ([207](#)) instance that is used when the service is in operation at runtime.

**See also:** TDaemonDef.DaemonClass ([216](#))

### **9.13.8 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 `TDaemon` 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.CreateDaemonInstance` (200) creates an instance of the daemon.

**See also:** `TDaemonDef.Instance` (216), `TDaemonDef.DaemonClass` (216), `RegisterDaemonClass` (197)

### **9.13.9 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` (217)

### **9.13.10 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.

### **9.13.11 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` (217)

### **9.13.12 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.

### 9.13.13 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.

### 9.13.14 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.

### 9.13.15 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 ([226](#))

### 9.13.16 TDaemonDef.OnCreateInstance

**Synopsis:** Event called when a daemon is instantiated

**Declaration:** Property OnCreateInstance : TNotifyEvent

**Visibility:** published

**Access:** Read,Write

**Description:** OnCreateInstance is called whenever an instance of the daemon is created. This can be used for instance when a single TDaemon class is used to run several services, to correctly initialize the TDaemon.

### 9.13.17 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: TDaemon.ReportStatus (207)

## 9.14 TDaemonDefs

### 9.14.1 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 (205) instance will create a collection and maintain it.

See also: TCustomDaemonMapper (205), TDaemonDef (215)

### 9.14.2 Method overview

Page	Property	Description
219	Create	Create a new instance of a TDaemonDefs collection.
220	DaemonDefByName	Find and return instance of daemon definition with given name.
220	FindDaemonDef	Find and return instance of daemon definition with given name.
220	IndexOfDaemonDef	Return index of daemon definition

### 9.14.3 Property overview

Page	Property	Access	Description
220	Daemons	rw	Indexed access to TDaemonDef instances

### 9.14.4 TDaemonDefs.Create

**Synopsis:** Create a new instance of a TDaemonDefs collection.

**Declaration:** constructor Create (AOwner: TPersistent; AClass: TCollectionItemClass)

**Visibility:** public

**Description:** Create creates a new instance of the TDaemonDefs collection. It keeps the AOwner parameter for future reference and calls the inherited constructor.

Normally it is not necessary to create an instance of TDaemonDefs manually. The global TCUSTOMDAEMONMAPPER (205) instance will create a collection and maintain it.

See also: TDaemonDef (215)

### **9.14.5 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 (215) instance with a name equal to DaemonName, and returns its index. It returns -1 if no definition was found with this name. The search is case insensitive.

**See also:** TDaemonDefs.FindDaemonDef (220), TDaemonDefs.DaemonDefByName (220)

### **9.14.6 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 (215) instance whose name matches DaemonName. If no definition is found, Nil is returned.

**See also:** TDaemonDefs.IndexOfDaemonDef (220), TDaemonDefs.DaemonDefByName (220)

### **9.14.7 TDaemonDefs.DaemonDefByName**

**Synopsis:** Find and return instance of daemon definition with given name.

**Declaration:** function DaemonDefByName(const DaemonName: string) : TDaemonDef

**Visibility:** public

**Description:** FindDaemonDef searches the list of daemon definitions and returns the TDaemonDef (215) instance whose name matches DaemonName. If no definition is found, an EDaemon (198) exception is raised.

The FindDaemonDef (220) call does not raise an error, but returns Nil instead.

**Errors:** If no definition is found, an EDaemon (198) exception is raised.

**See also:** TDaemonDefs.IndexOfDaemonDef (220), TDaemonDefs.FindDaemonDef (220)

### **9.14.8 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 (215)

## 9.15 TDaemonMapper

### 9.15.1 Description

TDaemonMapper is a direct descendent of TCustomDaemonMapper (205), 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` (221) constructor can be used instead.

See also: [CreateNew](#) (221), [Create](#) (221)

### 9.15.2 Method overview

Page	Property	Description
<a href="#">221</a>	<code>Create</code>	Create a new TDaemonMapper instance and initializes it from streamed resources.
<a href="#">221</a>	<code>CreateNew</code>	Create a new TDaemonMapper instance without initialization

### 9.15.3 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` (221) 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](#) (221)

### 9.15.4 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](#) (221)

## 9.16 TDaemonThread

### 9.16.1 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 (212) class will take care of this.

See also: TDaemonController (212), TDaemon (207)

### 9.16.2 Method overview

Page	Property	Description
<a href="#">223</a>	CheckControlMessage	Check if a control message has arrived
<a href="#">223</a>	ContinueDaemon	Continue the daemon
<a href="#">222</a>	Create	Create a new thread
<a href="#">222</a>	Execute	Run the daemon
<a href="#">224</a>	InterrogateDaemon	Report the daemon status
<a href="#">223</a>	PauseDaemon	Pause the daemon
<a href="#">224</a>	ShutDownDaemon	Shut down daemon
<a href="#">223</a>	StopDaemon	Stops the daemon

### 9.16.3 Property overview

Page	Property	Access	Description
<a href="#">224</a>	Daemon	r	Daemon instance

### 9.16.4 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 (224)

### 9.16.5 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 (207)

### 9.16.6 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.

### 9.16.7 TDaemonThread.StopDaemon

**Synopsis:** Stops the daemon

**Declaration:** function StopDaemon : Boolean; Virtual

**Visibility:** public

**Description:** StopDaemon attempts to stop the daemon using its TDaemon.Stop (207) method, and terminates the thread.

**See also:** TDaemon.Stop (207), TDaemonThread.PauseDaemon (223), TDaemonThread.ShutDownDaemon (224)

### 9.16.8 TDaemonThread.PauseDaemon

**Synopsis:** Pause the daemon

**Declaration:** function PauseDaemon : Boolean; Virtual

**Visibility:** public

**Description:** PauseDaemon attempts to stop the daemon using its TDaemon.Pause (207) method, and suspends the thread. It returns True if the attempt was successful.

**See also:** TDaemon.Pause (207), TDaemonThread.StopDaemon (223), TDaemonThread.ContinueDaemon (223), TDaemonThread.ShutDownDaemon (224)

### 9.16.9 TDaemonThread.ContinueDaemon

**Synopsis:** Continue the daemon

**Declaration:** function ContinueDaemon : Boolean; Virtual

**Visibility:** public

**Description:** ContinueDaemon attempts to stop the daemon using its TDaemon.Continue (207) method. It returns True if the attempt was successful.

**See also:** TDaemon.Continue (207), TDaemonThread.StopDaemon (223), TDaemonThread.PauseDaemon (223), TDaemonThread.ShutDownDaemon (224)

### 9.16.10 TDaemonThread.ShutDownDaemon

**Synopsis:** Shut down daemon

**Declaration:** function ShutDownDaemon : Boolean; Virtual

**Visibility:** public

**Description:** ShutDownDaemon shuts down the daemon. This happens normally only when the system is shut down and the daemon didn't respond to the stop request. The return result is the result of the TDaemon.Shutdown ([207](#)) function. The thread is terminated by this method.

**See also:** TDaemon.Shutdown ([207](#)), TDaemonThread.StopDaemon ([223](#)), TDaemonThread.PauseDaemon ([223](#)), TDaemonThread.ContinueDaemon ([223](#))

### 9.16.11 TDaemonThread.InterrogateDaemon

**Synopsis:** Report the daemon status

**Declaration:** function InterrogateDaemon : Boolean; Virtual

**Visibility:** public

**Description:** InterrogateDaemon simply calls TDaemon.ReportStatus ([207](#)) for the daemon that is running in this thread. It always returns True.

**See also:** TDaemon.ReportStatus ([207](#))

### 9.16.12 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 ([207](#))

## 9.17 TDependencies

### 9.17.1 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 ([225](#)) instances.

**See also:** TDependency ([225](#))

### 9.17.2 Method overview

Page	Property	Description
<a href="#">225</a>	Create	Create a new instance of a TDependencies collection.

### 9.17.3 Property overview

Page	Property	Access	Description
<a href="#">225</a>	Items	rw	Default property override

### 9.17.4 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.

### 9.17.5 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 ([225](#)).

**See also:** TDependency ([225](#))

## 9.18 TDependency

### 9.18.1 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 ([224](#)), TDaemonDef ([215](#))

### 9.18.2 Method overview

Page	Property	Description
<a href="#">226</a>	Assign	Assign TDependency instance to another

### 9.18.3 Property overview

Page	Property	Access	Description
<a href="#">226</a>	IsGroup	rw	Name refers to a service group
<a href="#">226</a>	Name	rw	Name of the service

### **9.18.4 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.

### **9.18.5 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 ([226](#))

### **9.18.6 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 ([226](#))

## **9.19 TWinBindings**

### **9.19.1 Description**

TWinBindings contains windows-specific properties for the daemon definition (in TDaemonDef.WinBindings ([218](#))). If the daemon should not run on Windows, then the properties can be ignored.

**See also:** TDaemonDef ([215](#)), TDaemonDef.WinBindings ([218](#))

### **9.19.2 Method overview**

Page	Property	Description
<a href="#">227</a>	Assign	Copies all properties
<a href="#">227</a>	Create	Create a new TWinBindings instance
<a href="#">227</a>	Destroy	Remove a TWinBindings instance from memory

### 9.19.3 Property overview

Page	Property	Access	Description
<a href="#">228</a>	Dependencies	rw	Service dependencies
<a href="#">227</a>	ErrCode	rw	Service specific error code
<a href="#">230</a>	ErrorSeverity	rw	Error severity in case of startup failure
<a href="#">228</a>	GroupName	rw	Service group name
<a href="#">230</a>	IDTag	rw	Location in the service group
<a href="#">229</a>	Password	rw	Password for service startup
<a href="#">230</a>	ServiceType	rw	Type of service
<a href="#">229</a>	StartType	rw	Service startup type.
<a href="#">229</a>	UserName	rw	Username to run service as
<a href="#">229</a>	WaitHint	rw	Timeout wait hint
<a href="#">228</a>	Win32ErrCode	rw	General windows error code

### 9.19.4 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 ([215](#)), TDaemonDef.WinBindings ([218](#)), TWinBindings.Dependencies ([228](#))

### 9.19.5 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 ([228](#)), TWinBindings.Create ([227](#))

### 9.19.6 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.

### 9.19.7 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 TDaemon.ReportStatus ([207](#)) to the windows service manager. If it is zero, then the contents of Win32ErrCode ([228](#)) are reported. If it is nonzero, then the windows-errorcode is set to ERROR\_SERVICE\_SPECIFIC\_ERROR.

See also: [TWinBindings.Win32ErrCode \(228\)](#)

### 9.19.8 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 TDaemon.ReportStatus ([207](#)) to the windows service manager. It is sent if ErrCode ([227](#)) is zero.

See also: [ErrCode \(227\)](#)

### 9.19.9 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.

### 9.19.10 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.ListGroup \(226\)](#)

### 9.19.11 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 \(229\)](#)

### 9.19.12 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 ([229](#)) property.

This property is only used during installation of the service.

See also: [Password \(229\)](#)

### 9.19.13 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.

### 9.19.14 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: [TDaemon.ReportStatus \(207\)](#)

### **9.19.15 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.

### **9.19.16 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.

### **9.19.17 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.

# Chapter 10

## Reference for unit 'db'

### 10.1 Used units

Table 10.1: Used units by unit 'db'

Name	Page
Classes	??
FmtBCD	??
MaskUtils	??
System	??
sysutils	??
Variants	??

### 10.2 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 (285) representing a set of records from a database, TField (334) which represents the contents of a field in a record, TDatasource (322) which acts as an event distributor on behalf of a dataset and TParams (407) which can be used to parametrize queries. The databases connections themselves are abstracted in the TDatabase (276) class.

### 10.3 Constants, types and variables

#### 10.3.1 Constants

```
DefaultFieldClasses : Array[TFIELDTYPE] of TFIELDCLASS = (Tfield, TStringField, TSma
```

DefaultFieldClasses contains the TField (334) descendent class to use when a TDataset instance needs to create fields based on the TFieldDefs (362) field definitions when opening the dataset. The entries can be set to create customized TField descendants for certain field datatypes in all datasets.

```
dsEditModes = [dsEdit, dsInsert, dsSetKey]
```

dsEditMode contains the various values of TDataset.State (312) for which the dataset is in edit mode, i.e. states in which it is possible to set field values for that dataset.

dsMaxBufferCount = MAXINT div 8

Maximum data buffers count for dataset

dsMaxStringSize = 8192

Maximum size of string fields

dsWriteModes = [dsEdit, dsInsert, dsSetKey, dsCalcFields, dsFilter, dsnewValue, dsIn

dsWriteModes contains the various values of TDataset.State (312) for which data can be written to the dataset buffer.

Fieldtypenames : Array[TFIELDTYPE] of string = ('Unknown', 'String', 'Smallint', 'In

FieldTypeNames contains the names (in english) for the various field data types.

FieldTypetoVariantMap : Array[TFIELDTYPE] of Integer = (varError, varOleStr, varSmal

FieldTypetoVariantMap contains for each field datatype the variant value type that corresponds to it. If a field type cannot be expressed by a variant type, then varError is stored in the variant value.

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

### 10.3.2 Types

LargeInt = Int64

Large (64-bit) integer

PBookmarkFlag = ^TBookmarkFlag

PBookmarkFlag is a convenience type, defined for internal use in TDataset (285) or one of its descendants.

PBufferList = ^TBufferList

PBufferList is a pointer to a structure of type TBufferList (234). 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 = AnsiString
```

TBlobData should never be used directly in application code.

```
TblobStreamMode = (bmRead, bmWrite, bmReadWrite)
```

Table 10.2: Enumeration values for type TBlobStreamMode

Value	Explanation
bmRead	Read blob data
bmReadWrite	Read and write blob data
bmWrite	Write blob data

TBlobStramMode is used when creating a stream for redaing BLOB data. It indicates what the data will be used for: reading, writing or both.

```
TblobType = ftBlob..ftWideMemo
```

TblobType is a subrange type, indicating the various datatypes of BLOB fields.

```
TBookmark = Pointer
```

TBookMark is the type used by the TDataset.SetBookMark ([285](#)) method. It is an opaque type, and should not be used any more, it is superseded by the TBookmarkStr ([234](#)) type.

```
TBookmarkFlag = (bfCurrent, bfBOF, bfEOF, bfInserted)
```

Table 10.3: Enumeration values for type TBookmarkFlag

Value	Explanation
bfBOF	First record in the dataset.
bfCurrent	Buffer used for the current record
bfEOF	Last record in the dataset
bfInserted	Buffer used for insert

TBookmarkFlag is used internally by TDataset ([285](#)) and it's descendent types to mark the internal memory buffers. It should not be used in end-user applications.

---

```
TBookmarkStr = String deprecated
```

`TBookmarkStr` is the type used by the `TDataset.Bookmark` ([306](#)) 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` ([285](#)) 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 10.4: Enumeration values for type `TDataAction`

Value	Explanation
<code>daAbort</code>	The operation should be aborted (edits are undone, and an <code>EAbort</code> exception is raised)
<code>daFail</code>	The operation should fail (an exception will be raised)
<code>daRetry</code>	Retry the operation.

`TDataAction` is used by the `TDatasetErrorEvent` ([235](#)) 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` ([276](#)) class.

```
TDataChangeEvent = procedure(Sender: TObject;Field: TField) of object
```

`TDataChangeEvent` is the event handler prototype for the `TDatasource.OnDataChange` ([325](#)) 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 10.5: Enumeration values for type TDataEvent

Value	Explanation
deCheckBrowseMode	The browse mode is being checked
deConnectChange	Unused
deDataSetChange	The dataset property changed
deDataSetScroll	The dataset scrolled to another record
deDisabledStateChange	Unused
deFieldChange	A field value changed
deFieldListChange	Event sent when the list of fields of a dataset changes
deFocusControl	Event sent whenever a control connected to a field should be focused
deLayoutChange	The layout properties of one of the fields changed
deParentScroll	Unused
dePropertyChange	Unused
deReconcileError	Unused
deRecordChange	The current record changed
deUpdateRecord	The record is being updated
deUpdateState	The dataset state is updated

TDataEvent describes the various events that can be sent to TDatasource (322) instances connected to a TDataset (285) 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 (285) 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 (321), TDataset.OnPostError (322) and TDataset.OnDeleteError (320) 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 its 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 (234).

```
TDataSetNotifyEvent = procedure(DataSet: TDataSet) of object
```

TDataSetNotifyEvent is used in most of the TDataset (285) 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)
```

Table 10.6: Enumeration values for type TDataSetState

Value	Explanation
dsBlockRead	The dataset is open, but no events are transferred to datasources.
dsBrowse	The dataset is active, and the cursor can be used to navigate the data.
dsCalcFields	The dataset is calculating it's calculated fields.
dsCurValue	The dataset is showing the current values of a record.
dsEdit	The dataset is in editing mode: the current record can be modified.
dsFilter	The dataset is filtering records.
dsInactive	The dataset is not active. No data is available.
dsInsert	The dataset is in insert mode: the current record is a new record which can be edited.
dsInternalCalc	The dataset is calculating it's internally calculated fields.
dsNewValue	The dataset is showing the new values of a record.
dsOldValue	The dataset is showing the old values of a record.
dsOpening	The dataset is currently opening, but is not yet completely open.
dsSetKey	The dataset is calculating the primary key.

TDataSetState describes the current state of the dataset. During it's lifetime, the dataset's state is described by these enumerated values.

Some state are not used in the default TDataset implementation, and are only used by certain descendants.

```
TDatetimeAlias = TDatetime
```

TDatetimeAlias is no longer used.

```
TDatetimeRec = record
end
```

TDatetimeRec was used by older TDataset (285) 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 (328)

```
TDBTransactionClass = Class of TDBTransaction
```

TDBTransactionClass is the class pointer for the TDBTransaction (329) class.

```
TFieldAttribute = (faHiddenCol, faReadonly, faRequired, faLink, faUnNamed,
                   faFixed)
```

Table 10.7: Enumeration values for type TFieldAttribute

Value	Explanation
faFixed	Fixed length field
faHiddenCol	Field is a hidden column (used to construct a unique key)
faLink	Field is a link field for other datasets
faReadOnly	Field is read-only
faRequired	Field is required
faUnNamed	Field has no original name

TFieldAttribute is used to denote some attributes of a field in a database. It is used in the Attributes (361) property of TFieldDef (358).

TFieldAttributes = Set of TFieldAttribute

TFieldAttributes is used in the TFieldDef.Attributes (361) property to denote additional attributes of the underlying field.

TFieldChars = Set of Char

TFieldChars is a type used in the TField.ValidChars (350) property. It's a simple set of characters.

TFieldClass = Class of TField

```
TFieldGetTextEvent = procedure(Sender: TField; var aText: string;
                               DisplayText: Boolean) of object
```

TFieldGetTextEvent is the prototype for the TField.OnGetText (358) 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 10.8: Enumeration values for type TFieldKind

Value	Explanation
fkCalculated	The field is calculated on the fly.
fkData	Field represents actual data in the underlying data structure.
fkInternalCalc	Field is calculated but stored in an underlying buffer.
fkLookup	The field is a lookup field.

TFieldKind indicates the type of a TField instance. Besides TField instances that represent fields present in the underlying data records, there can also be calculated or lookup fields. To distinguish between these kind of fields, TFieldKind is introduced.

TFieldKinds = Set of TFieldKind

TFieldKinds is a set of TFieldKind ([237](#)) 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 ([334](#)) class. Its Sender parameter is the field instance that triggered the event.

```
TFieldRef = ^TField
```

Pointer to a TField instance

```
TFieldSetTextEvent = procedure(Sender: TField; const aText: string)
                     of object
```

TFieldSetTextEvent is the prototype for an event handler used to set the contents of a field based on a user-edited text. It should be used when the text of a field is entered with special formatting. The event handler should set the contents of the field based on the formatted text in the AText parameter.

```
TFieldType = (ftUnknown, ftString, ftSmallint, ftInteger, ftWord, ftBoolean,
               ftFloat, ftCurrency, ftBCD, ftDate, ftTime, ftDateTime, ftBytes,
               ftVarBytes, ftAutoInc, ftBlob, ftMemo, ftGraphic, ftFmtMemo,
               ftParadoxOle, ftDBaseOle, ftTypedBinary, ftCursor,
               ftFixedChar, ftWideString, ftLargeint, ftADT, ftArray,
               ftReference, ftDataSet, ftOraBlob, ftOraClob, ftVariant,
               ftInterface, ftIDispatch, ftGuid, ftTimeStamp, ftFMTBcd,
               ftFixedWideChar, ftWideMemo)
```

Table 10.9: Enumeration values for type TFieldType

Value	Explanation
ftADT	ADT value
ftArray	Array data
ftAutoInc	Auto-increment integer value (4 bytes)
ftBCD	Binary Coded Decimal value (DECIMAL and NUMERIC SQL types)
ftBlob	Binary data value (no type, no size)
ftBoolean	Boolean value
ftBytes	Array of bytes value, fixed size (untyped)
ftCurrency	Currency value (4 decimal points)
ftCursor	Cursor data value (no size)
ftDataSet	Dataset data (blob)
ftDate	Date value
ftDateTime	Date/Time (timestamp) value
ftDBaseOle	Paradox OLE field data
ftFixedChar	Fixed character array (string)
ftFixedWideChar	Fixed wide character date (2 bytes per character)
ftFloat	Floating point value (double)
ftFMTBcd	Formatted BCD (Binary Coded Decimal) value.
ftFmtMemo	Formatted memo ata value (no size)
ftGraphic	Graphical data value (no size)
ftGuid	GUID data value
ftIDispatch	Dispatch data value
ftInteger	Regular integer value (4 bytes, signed)
ftInterface	interface data value
ftLargeint	Large integer value (8-byte)
ftMemo	Binary text data (no size)
ftOraBlob	Oracle BLOB data
ftOraClob	Oracle CLOB data
ftParadoxOle	Paradox OLE field data (no size)
ftReference	Reference data
ftSmallint	Small integer value(1 byte, signed)
ftString	String data value (ansistring)
ftTime	Time value
ftTimeStamp	Timestamp data value
ftTypedBinary	Binary typed data (no size)
ftUnknown	Unknown data type
ftVarBytes	Array of bytes value, variable size (untyped)
ftVariant	Variant data value
ftWideMemo	Widestring memo data
ftWideString	Widestring (2 bytes per character)
ftWord	Word-sized value(2 bytes, unsigned)

TFieldType indicates the type of a TField (334) underlying data, in the DataType (346) property.

```
TFilterOption = (foCaseInsensitive, foNoPartialCompare)
```

Table 10.10: Enumeration values for type TFILTEROPTION

Value	Explanation
foCaseInsensitive	Filter case insensitively.
foNoPartialCompare	Do not compare values partially, always compare completely.

TFILTEROPTION enumerates the various options available when filtering a dataset. The TFILTEROPTIONS (240) set is used in the TDataset.FilterOptions (313) 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 (313) property. The available values are described in the TFILTEROPTION (239) type.

```
TFILTERRECORDEVENT = procedure(DataSet: TDataSet; var Accept: Boolean)
of object
```

TFILTERRECORDEVENT is the prototype for the TDataset.OnFilterRecord (321) 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 10.11: Enumeration values for type TGETMODE

Value	Explanation
gmCurrent	Retrieve the current record
gmNext	Retrieve the next record.
gmPrior	Retrieve the previous record.

TGETMODE is used internally by TDataset (285) 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 10.12: Enumeration values for type TGETRESULT

Value	Explanation
grBOF	The beginning of the recordset is reached
grEOF	The end of the recordset is reached.
grError	An error occurred
grOK	The operation was completed successfully

TGETRESULT is used by descendants of TDataset (285) when they have to communicate the result of the GetRecord operation back to the TDataset record.

---

```
TIndexOption = (ixPrimary, ixUnique, ixDescending, ixCaseInsensitive,
                 ixExpression, ixNonMaintained)
```

Table 10.13: Enumeration values for type TIndexOption

Value	Explanation
ixCaseInsensitive	The values in the index are sorted case-insensitively
ixDescending	The values in the index are sorted descending.
ixExpression	The values in the index are based on a calculated expression.
ixNonMaintained	The index is non-maintained, i.e. changing the data will not update the index.
ixPrimary	The index is the primary index for the data
ixUnique	The index is a unique index, i.e. each index value can occur only once

TIndexOption describes the various properties that an index can have. It is used in the TIndexOptions (241) set type to describe all properties of an index definition as in TIndexDef (378).

```
TIndexOptions = Set of TIndexOption
```

TIndexOptions contains the set of properties that an index can have. It is used in the TIndexDef.Options (380) property to describe all properties of an index definition as in TIndexDef (378).

```
TLocateOption = (loCaseInsensitive, loPartialKey)
```

Table 10.14: Enumeration values for type TLocateOption

Value	Explanation
loCaseInsensitive	Perform a case-insensitive search
loPartialKey	Accept partial key matches for string fields

TLocateOption is used in the TDataset.Locate (301) 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 (301) 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 a the TCustomConnection.OnLogin (275) event handler. It gets passed the TCustomConnection instance that is trying to login, and the initial username and password.

```
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.

```
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.

```
TParamStyle = (psInterbase,psPostgreSQL,psSimulated)
```

Table 10.15: Enumeration values for type `TParamStyle`

Value	Explanation
<code>psInterbase</code>	Parameters are specified by a ? character
<code>psPostgreSQL</code>	Parameters are specified by a \$N character.
<code>psSimulated</code>	Parameters are specified by a \$N character.

`TParamStyle` denotes the style in which parameters are specified in a query. It is used in the `TParams.ParseSQL` (410) 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 10.16: Enumeration values for type `TParamType`

Value	Explanation
<code>ptInput</code>	Input parameter
<code>ptInputOutput</code>	Input/output parameter
<code>ptOutput</code>	Output parameter, filled on result
<code>ptResult</code>	Result parameter
<code>ptUnknown</code>	Unknown type

`TParamType` indicates the kind of parameter represented by a `TParam` (395) 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 (242) values.

```
TProviderFlag = (pfInUpdate, pfInWhere, pfInKey, pfHidden)
```

Table 10.17: Enumeration values for type TProviderFlag

Value	Explanation
pfHidden	
pfInKey	Field is a key field and used in the WHERE clause of an update statement
pfInUpdate	Changes to the field should be propagated to the database.
pfInWhere	Field should be used in the WHERE clause of an update statement in case of upWhereChanged.

TProviderFlag describes how the field should be used when applying updates from a dataset to the database. Each field of a TDataset (285) has one or more of these flags.

```
TProviderFlags = Set of TProviderFlag
```

TProviderFlags is used for the TField.ProviderFlags (356) 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 10.18: Enumeration values for type TPSCommandType

Value	Explanation
ctDDL	SQL DDL statement
ctDelete	SQL DELETE Statement
ctInsert	SQL INSERT Statement
ctQuery	General SQL statement
ctSelect	SQL SELECT Statement
ctStoredProc	Stored procedure statement
ctTable	Table contents (select * from table)
ctUnknown	Unknown SQL type or not SQL based
ctUpdate	SQL UPDATE statement

TPSCommandType is used in the IProviderSupport.PSGetCommandType (252) 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` (285) 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` (244)

```
TResolverResponse = (rrSkip, rrAbort, rrMerge, rrApply, rrIgnore)
```

Table 10.19: Enumeration values for type `TResolverResponse`

Value	Explanation
rrAbort	Abort the whole update process
rrApply	Replace the update with new values applied by the event handler
rrIgnore	Ignore the error and remove update from change log
rrMerge	Merge the update with existing changes on the server
rrSkip	Skip the current update, leave it in the change log

`TResolverResponse` is used to indicate what should happen to a pending change that could not be resolved. It is used in callbacks.

```
TResyncMode= Set of (rmExact, rmCenter)
```

Table 10.20: Enumeration values for type

Value	Explanation
rmCenter	Try to position the cursor in the middle of the buffer
rmExact	Reposition at exact the same location in the buffer

`TResyncMode` is used internally by various `TDataset` (285) navigation and data manipulation methods such as the `TDataset.Refresh` (304) method when they need to reset the cursor position in the dataset's buffer.

```
TStringFieldBuffer = Array[0..dsMaxStringSize] of Char
```

Type to access string field content buffers as an array of characters

```
TUpdateAction = (uaFail, uaAbort, uaSkip, uaRetry, uaApplied)
```

Table 10.21: Enumeration values for type `TUpdateAction`

Value	Explanation
uaAbort	The whole update operation should abort
uaApplied	Consider the update as applied
uaFail	Update operation should fail
uaRetry	Retry the update operation
uaSkip	The update of the current record should be skipped. (but not discarded)

TUpdateAction indicates what action must be taken in case the applying of updates on the underlying database fails. This type is not used in the TDataset (285) class, but is defined on behalf of TDataset descendants 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 10.22: Enumeration values for type TUpdateKind

Value	Explanation
ukDelete	Delete a record in the database.
ukInsert	insert a new record in the database.
ukModify	Modify an existing record in the database.

TUpdateKind indicates what kind of update operation is in progress when applying updates.

```
TUpdateMode = (upWhereAll,upWhereChanged,upWhereKeyOnly)
```

Table 10.23: Enumeration values for type TUpdateMode

Value	Explanation
upWhereAll	Use all old field values
upWhereChanged	Use only old field values of modified fields
upWhereKeyOnly	Only use key fields in the where clause.

TUpdateMode determines how the WHERE clause of update queries for SQL databases should be constructed.

```
TUpdateStatus = (usUnmodified,usModified,usInserted,usDeleted)
```

Table 10.24: Enumeration values for type TUpdateStatus

Value	Explanation
usDeleted	Record exists in the database, but is locally deleted.
usInserted	Record does not yet exist in the database, but is locally inserted
usModified	Record exists in the database but is locally modified
usUnmodified	Record is unmodified

TUpdateStatus determines the current state of the record buffer, if updates have not yet been applied to the database.

```
TUpdateStatusSet = Set of TUpdateStatus
```

TUpdateStatusSet is a set of TUpdateStatus (245) values.

## 10.4 Procedures and functions

### 10.4.1 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 (??)

### 10.4.2 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 (248) exception, passing it Msg. If Comp is specified, the name of the component is prepended to the message.

**See also:** DatabaseErrorFmt (246), EDatabaseError (248)

### 10.4.3 DatabaseErrorFmt

**Synopsis:** Raise an EDatabaseError exception with a formatted message

**Declaration:** procedure DatabaseErrorFmt(const Fmt: string; Args: Array of const); Overload  
procedure DatabaseErrorFmt(const Fmt: string; Args: Array of const; Comp: TComponent); Overload

**Visibility:** default

**Description:** DatabaseErrorFmt raises an EDatabaseError (248) 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 (246), EDatabaseError (248)

### 10.4.4 DateTimeRecToDateTIme

**Synopsis:** Convert TDateTimeRec record to a TDateTime value.

**Declaration:** function DateTimeRecToDateTIme(DT: TFieldType; Data: TDateTimeRec): TDateTime

Visibility: default

Description: `DateTimeRecToDate` examines `Data` and `Dt` and uses `dt` to convert the timestamp in `Data` to a `TDateTime` value.

See also: [TFieldType \(238\)](#), [TDateTimeRec \(236\)](#), [DateTimeToDateRec \(247\)](#)

#### 10.4.5 **DateTimeToDateRec**

Synopsis: Convert `TDateTime` value to a `TDateTimeRec` record.

Declaration: `function DateTimeToDateRec(DT: TFieldType; Data: TDateTime) : TDateTimeRec`

Visibility: default

Description: `DateTimeToDateRec` examines `Data` and `Dt` and uses `dt` to convert the date/time value in `Data` to a `TDateTimeRec` record.

See also: [TFieldType \(238\)](#), [TDateTimeRec \(236\)](#), [DateTimeRecToDate \(246\)](#)

#### 10.4.6 **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 (??)`

#### 10.4.7 **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 \(365\)](#)

### 10.4.8 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 (410)

## 10.5 EDatabaseError

### 10.5.1 Description

EDatabaseError is the base class from which database-related exception classes should derive. It is raised by the DatabaseError (246) call.

See also: DatabaseError (246), DatabaseErrorFmt (246)

## 10.6 EUpdateError

### 10.6.1 Description

EupdateError is an exception used by the TProvider database support. It should never be raised directly.

See also: EDatabaseError (248)

### 10.6.2 Method overview

Page	Property	Description
249	Create	Create a new EUpdateError instance
249	Destroy	Free the EupdateError instance

### 10.6.3 Property overview

Page	Property	Access	Description
249	Context	r	Context in which exception occurred.
249	ErrorCode	r	Numerical error code.
250	OriginalException	r	Originally raised exception
250	PreviousError	r	Previous error number

#### **10.6.4 EUpdateError.Create**

**Synopsis:** Create a new EUpdateError instance

**Declaration:** constructor Create(NativeError: string; Context: string; ErrCode: Integer; PrevError: Integer; E: Exception);

**Visibility:** public

**Description:** Create instantiates a new EUpdateError object and populates the various properties with the NativeError, Context, ErrCode and PrevError parameters. The E parameter is the actual exception that occurred while the update operation was attempted. The exception object E will be freed if the EUpdateError instance is freed.

**See also:** [EDatabaseError \(248\)](#)

#### **10.6.5 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 \(250\)](#)

#### **10.6.6 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 \(250\)](#), [EUpdateError.ErrorCode \(249\)](#), [EUpdateError.PreviousError \(250\)](#)

#### **10.6.7 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 \(250\)](#), [EUpdateError.Context \(249\)](#), [EUpdateError.PreviousError \(250\)](#)

### 10.6.8 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 (231)

### 10.6.9 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 (249), EUpdateError.Context (249), EUpdateError.OriginalException (250)

## 10.7 IProviderSupport

### 10.7.1 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 (285) 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 (285)

### 10.7.2 Method overview

Page	Property	Description
251	PSEndTransaction	End an active transaction
251	PSEExecute	Execute the current command-text.
251	PSEExecuteStatement	Execute a SQL statement.
252	PSGetAttributes	Get a list of attributes (metadata)
252	PSGetCommandText	Return the SQL command executed for getting data.
252	PSGet CommandType	Return SQL command type
253	PSGetDefaultOrder	Default order index definition
253	PSGetIndexDefs	Return a list of index definitions
253	PSGetKeyFields	Return a list of key fields in the dataset
253	PSGetParams	Get the parameters in the commandtext
254	PSGetQuoteChar	Quote character for quoted strings
254	PSGetTableName	Name of database table which must be updated
254	PSGetUpdateException	Transform exception to UpdateError
254	PSInTransaction	Is the dataset in an active transaction.
255	PSIsSQLBased	Is the dataset SQL based
255	PSIsSQLSupported	Can the dataset support SQL statements
255	PSReset	Position the dataset on the first record
255	PSSetCommandText	Set the command-text of the dataset
256	PSSetParams	Set the parameters for the command text
256	PSStartTransaction	Start a new transaction
256	PSUpdateRecord	Update a record

### 10.7.3 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 (231) returns True). If ACommit is True then the transaction is committed, else it is rollbacked.

See also: PSInTransaction (231), PSStartTransaction (231)

### 10.7.4 IProviderSupport.PSEExecute

Synopsis: Execute the current command-text.

Declaration: procedure PSEExecute

Visibility: default

Description: PSEExecute executes the current SQL statement: the command as it is returned by PSGetCommandText (231).

See also: PSGetCommandText (231), PSEExecuteStatement (231)

### 10.7.5 IProviderSupport.PSEExecuteStatement

Synopsis: Execute a SQL statement.

**Declaration:** 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 (231) returns the same value before and after a call to PSEExecuteStatement.

**See also:** PSGetCommandText (231), PSSetCommandText (231), PSEExecuteStatement (231)

### 10.7.6 IProviderSupport.PSGetAttributes

**Synopsis:** Get a list of attributes (metadata)

**Declaration:** procedure PSGetAttributes(List: TList)

**Visibility:** default

**Description:** PSGetAttributes returns a set of name=value pairs which is included in the data packet sent to a client.

**See also:** PSGetCommandText (231)

### 10.7.7 IProviderSupport.PSGetCommandText

**Synopsis:** Return the SQL command executed for getting data.

**Declaration:** function PSGetCommandText : string

**Visibility:** default

**Description:** PSGetCommandText returns the SQL command that is executed when the PSEExecute (231) function is called (for a TSQLQuery this would be the SQL property) or when the dataset is opened.

**See also:** PSEExecute (231), PSSetCommandText (231)

### 10.7.8 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 (231)). The list of possible command types is enumerated in TPSCommandType (243).

**See also:** PSGetCommandText (231), TPSCommandType (243), PSEExecute (231)

### **10.7.9 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 (231)) that represents the default sort order.

**See also:** PSGetIndexDefs (231), PSGetKeyFields (231)

### **10.7.10 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 (231), PSGetKeyFields (231)

### **10.7.11 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 (334) property.

**See also:** PSGetIndexDefs (231), PSGetDefaultOrder (231), TField.ProviderOptions (334), TProviderFlags (243)

### **10.7.12 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 (231)). This is usually the Params property of a TDataset (285) descendant.

**See also:** PSGetCommandText (231), PSSetParams (231)

### 10.7.13 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 (231)

### 10.7.14 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 (231)

### 10.7.15 IProviderSupport.PSGetUpdateException

**Synopsis:** Transform exception to EUpdateError

**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 (248) exception. The previous EUpdateError exception in the update batch is passed in Prev.

See also: EUpdateError (248)

### 10.7.16 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 (231), PSStartTransaction (231)

### 10.7.17 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 (231) which indicates whether SQL statements can be executed using PSEExecuteCommand (231)

**See also:** PSIsSQLSupported (231), PSEExecuteCommand (231)

### 10.7.18 IProviderSupport.PSIsSQLSupported

**Synopsis:** Can the dataset support SQL statements

**Declaration:** function PSIsSQLSupported : Boolean

**Visibility:** default

**Description:** PSIsSQLSupported returns True if PSEExecuteCommand (231) can be used to execute SQL statements on the underlying database.

**See also:** PSEExecuteCommand (231)

### 10.7.19 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 (298), TDataset.Open (303)

### 10.7.20 IProviderSupport.PSSetCommandText

**Synopsis:** Set the command-text of the dataset

**Declaration:** procedure PSSetCommandText (const CommandText: string)

**Visibility:** default

**Description:** PSSetCommandText sets the commandtext (SQL) statement that is executed by PSEExecute or that is used to open the dataset.

**See also:** PSEExecute (231), PSGetCommandText (231), PSSetParams (231)

### 10.7.21 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 (231), PSGetParams (231)

### 10.7.22 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 (231) returned False).

**See also:** PSEndTransaction (231), PSIntransaction (231)

### 10.7.23 IProviderSupport.PSUpdateRecord

**Synopsis:** Update a record

**Declaration:** function PSUpdateRecord(UpdateKind: TUpdateKind; Delta: TDataSet)  
                  : Boolean

**Visibility:** default

**Description:** PSUpdateRecord is called before attempting to update the records through generated SQL statements. The update to be performed is passed in UpdateKind parameter. The Delta Dataset's current record contains all data for the record that must be updated.

The function returns True if the update was successfully applied, False if not. In that case the provider will attempt to update the record using SQL statements if the dataset allows it.

**See also:** PSIsSQLSupported (231), PSEExecuteCommand (231)

## 10.8 TAutoIncField

### 10.8.1 Description

TAutoIncField is the class created when a dataset must manage 32-bit signed integer data, of datatype ftAutoInc: This field gets its 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 (334)

### 10.8.2 Method overview

Page	Property	Description
<a href="#">257</a>	Create	Create a new instance of the <code>TAutoIncField</code> class.

### 10.8.3 `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` ([334](#)) class' fields.

See also: `TField` ([334](#))

## 10.9 `TBCDField`

### 10.9.1 Description

`TBCDField` is the class used when a dataset must manage data of Binary Coded Decimal type. (`TField.DataType` ([346](#)) equals `ftBCD`). It initializes some of the properties of the `TField` ([334](#)) 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` ([374](#)) 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` ([285](#)), `TField` ([334](#)), `TFMTBCDField` ([374](#))

### 10.9.2 Method overview

Page	Property	Description
<a href="#">258</a>	CheckRange	Check whether a values falls within the allowed range
<a href="#">257</a>	Create	Create a new instance of a <code>TBCDField</code> class.

### 10.9.3 Property overview

Page	Property	Access	Description
<a href="#">259</a>	Currency	rw	Does the field represent a currency amount
<a href="#">259</a>	MaxValue	rw	Maximum value for the field
<a href="#">259</a>	MinValue	rw	Minimum value for the field
<a href="#">258</a>	Precision	rw	Precision of the BCD field
<a href="#">260</a>	Size		Number of decimals after the decimal separator
<a href="#">258</a>	Value	rw	Value of the field contents as a Currency type

### 10.9.4 `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 (334) properties to configure the instance for working with BCD data values.

**See also:** TField (334)

### **10.9.5 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 (259) and MaxValue (259) properties. If the value lies outside of the allowed range, then False is returned.

**See also:** MaxValue (259), MinValue (259)

### **10.9.6 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 (350) property to a currency type field. It returns the same value as the TField.AsCurrency (341) field.

**See also:** TField.Value (350), TField.AsCurrency (341)

### **10.9.7 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 (260), 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 (260), TBCDField.Value (258)

### **10.9.8 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 (346) and TField.Text (349) 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 (346), TField.Text (349)

### **10.9.9 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 (259) should also be set, because it will also be checked.

**See also:** TBCDField.MinValue (259), TBCDField.CheckRange (258)

### **10.9.10 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 (259) should also be set, because it will also be checked.

**See also:** TBCDField.MaxValue (259), TBCDField.CheckRange (258)

### 10.9.11 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 ([258](#)) field.

**See also:** TBCDField.Precision ([258](#))

## 10.10 TBinaryField

### 10.10.1 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 ([334](#)) 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 ([266](#)) or TVarBytesField ([416](#)) should be created.

**See also:** TDataset ([285](#)), TField ([334](#)), TBytesField ([266](#)), TVarBytesField ([416](#))

### 10.10.2 Method overview

Page	Property	Description
<a href="#">260</a>	Create	Create a new instance of a TBinaryField class.

### 10.10.3 Property overview

Page	Property	Access	Description
<a href="#">261</a>	Size		Size of the binary data

### 10.10.4 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 ([334](#))

### 10.10.5 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 \(349\)](#)

## 10.11 TBlobField

### 10.11.1 Description

TBlobField is the class used when a dataset must manage BLOB data. ([TField.DataType \(346\)](#) equals `ftBLOB`). It initializes some of the properties of the [TField \(334\)](#) 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 \(392\)](#), [TWideMemoField \(417\)](#) or [TGraphicField \(376\)](#).

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 \(285\)](#), [TField \(334\)](#), [TMemoField \(392\)](#), [TWideMemoField \(417\)](#), [TGraphicField \(376\)](#)

### 10.11.2 Method overview

Page	Property	Description	
262	Clear	Clear the BLOB field's contents	
261	Create	Create a new instance of a <a href="#">TBlobField</a> class.	
262	IsBlob	Is the field a blob field	
262	LoadFromFile	Load the contents of the field from a file	
262	LoadFromStream	Load the field's contents from stream	
263	SaveToFile	Save field contents to a file	
263	SaveToStream	Save the field's contents to stream	
263	SetFieldType	Set field type	

### 10.11.3 Property overview

Page	Property	Access	Description
263	BlobSize	r	Size of the current blob
265	BlobType	rw	Type of blob
264	Modified	rw	Has the field's contents been modified.
265	Size		Size of the blob field
264	Transliterate	rw	Should the contents of the field be transliterated
264	Value	rw	Return the field's contents as a string

### 10.11.4 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 (334) properties to configure the instance for working with BLOB data.

See also: TField (334)

### **10.11.5 TBlobField.Clear**

Synopsis: Clear the BLOB field's contents

Declaration: procedure Clear;   Override

Visibility: public

Description: Clear overrides the TField implementation of TField.Clear (338). It creates and immediately releases an empty blob stream in write mode, effectively clearing the contents of the BLOB field.

See also: TField.Clear (338), TField.IsNull (348)

### **10.11.6 TBlobField.IsBlob**

Synopsis: Is the field a blob field

Declaration: class function IsBlob;   Override

Visibility: public

Description: IsBlob is overridden by TBlobField to return True

See also: TField.IsBlob (339)

### **10.11.7 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 (262) 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 (262), SaveToFile (263)

### **10.11.8 TBlobField.LoadFromStream**

Synopsis: Load the field's contents from stream

Declaration: procedure LoadFromStream(Stream: TStream)

Visibility: public

Description: LoadFromStream can be used to load the contents of the field from a TStream (??) descendent. The entire data of the stream will be copied, and the stream will be positioned on the first byte of data, so it must be seekable.

**Errors:** If the stream is not seekable, an exception will be raised.

**See also:** [SaveToStream \(263\)](#), [LoadFromFile \(262\)](#)

### **10.11.9 TBlobField.SaveToFile**

**Synopsis:** Save field contents to a file

**Declaration:** procedure SaveToFile(const FileName: string)

**Visibility:** public

**Description:** SaveToFile creates a file stream with FileName as the name of the file to open, en then calls [SaveToStream \(263\)](#) 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 \(262\)](#), [SaveToStream \(263\)](#)

### **10.11.10 TBlobField.SaveToStream**

**Synopsis:** Save the field's contents to stream

**Declaration:** procedure SaveToStream(Stream: TStream)

**Visibility:** public

**Description:** SaveToStream can be used to save the contents of the field to a TStream (??) descendent. The entire data of the field will be copied. The stream must of course support writing.

**Errors:** If the stream is not writable, an exception will be raised.

**See also:** [SaveToFile \(263\)](#), [LoadFromStream \(262\)](#)

### **10.11.11 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 \(346\)](#)

### **10.11.12 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 \(349\)](#), [TField.DataSize \(346\)](#)

### 10.11.13 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 \(262\)](#)

### 10.11.14 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 (343)` property.

**See also:** [TField.Value \(350\)](#), [TField.AsString \(343\)](#)

### 10.11.15 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 (305)` 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 \(414\)](#), [TDataset.Translate \(305\)](#)

### 10.11.16 TBlobField.BlobType

**Synopsis:** Type of blob

**Declaration:** Property BlobType : TBlobType

**Visibility:** published

**Access:** Read,Write

**Description:** BlobType is an alias for TField.DataType (346), but with a restricted set of values. Setting BlobType is equivalent to setting the TField.DataType (346) property.

**See also:** TField.DataType (346)

### 10.11.17 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 (263) property instead.

**See also:** BlobSize (263)

## 10.12 TBooleanField

### 10.12.1 Description

TBooleanField is the field class used by TDataset (285) whenever it needs to manage boolean data (TField.DataType (346) equals ftBoolean). It overrides some properties and methods of TField (334) 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 (285), TField (334)

### 10.12.2 Method overview

Page	Property	Description
266	Create	Create a new instance of the TBooleanField class.

### 10.12.3 Property overview

Page	Property	Access	Description
266	DisplayValues	rw	Textual representation of the true and false values
266	Value	rw	Value of the field as a boolean value

### 10.12.4 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 (334) properties to configure it for working with boolean values.

**See also:** TField (334)

### 10.12.5 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 (350) by TBooleanField as a boolean value. It returns the same value as the TField.AsBoolean (341) property.

**See also:** TField.AsBoolean (341), TField.Value (350)

### 10.12.6 TBooleanField.DisplayValues

**Synopsis:** Textual representation of the true and false values

**Declaration:** Property DisplayValues : string

**Visibility:** published

**Access:** Read,Write

**Description:** DisplayValues contains 2 strings, separated by a semicolon (;) which are used to display the True and False values of the fields. The first string is used for True values, the second value is used for False values. If only one value is given, it will serve as the representation of the True value, the False value will be represented as an empty string.

A value of Yes;No will result in True values being displayed as 'Yes', and False values as 'No'. When writing the value of the field as a string, the string will be compared (case insensitively) with the value for True, and if it matches, the field's value will be set to True. After this it will be compared to the value for False, and if it matches, the field's value will be set to False. If the text matches neither of the two values, an exception will be raised.

**See also:** TField.AsString (343), TField.Text (349)

## 10.13 TBytesField

### 10.13.1 Description

TBytesField is the class used when a dataset must manage data of fixed-size binary type. (TField.DataType (346) equals ftBytes). It initializes some of the properties of the TField (334) 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 \(285\)](#), [TField \(334\)](#), [TVarBytesField \(416\)](#)

### 10.13.2 Method overview

Page	Property	Description
<a href="#">267</a>	Create	Create a new instance of a <code>TBytesField</code> class.

### 10.13.3 `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` ([334](#)) properties to configure the instance for working with binary data values.

See also: [TField \(334\)](#)

## 10.14 TCheckConstraint

### 10.14.1 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` ([269](#)) property , or a constraint enforced by the user on a particular dataset instance stored in `TCheckConstraint.CustomConstraint` ([268](#))

See also: [TCheckConstraints \(269\)](#), [TCheckConstraint.ImportedConstraint](#) ([269](#)), [TCheckConstraint.CustomConstraint](#) ([268](#))

### 10.14.2 Method overview

Page	Property	Description
<a href="#">268</a>	Assign	Assign one constraint to another

### 10.14.3 Property overview

Page	Property	Access	Description
<a href="#">268</a>	CustomConstraint	rw	User-defined constraint
<a href="#">268</a>	ErrorMessage	rw	Message to display when the constraint is violated
<a href="#">268</a>	FromDictionary	rw	True if the constraint is imported from a datadictionary
<a href="#">269</a>	ImportedConstraint	rw	Constraint imported from the database engine

#### 10.14.4 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 (269), TCheckConstraint.CustomConstraint (268)

#### 10.14.5 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 (285) 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 (268)

**See also:** TCheckConstraint(ErrorMessage (268)

#### 10.14.6 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 (268)

#### 10.14.7 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 (285) descendants to indicate the source of the constraint, but is otherwise ignored.

**See also:** TCheckConstraint.ImportedConstraint (269)

### 10.14.8 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 (285) descendent.

See also: TCheckConstraint.CustomConstraint (268)

## 10.15 TCheckConstraints

### 10.15.1 Description

TCheckConstraints is a TCollection descendent which keeps a collection of TCheckConstraint (267) items. It overrides the Add (270) method to return a TCheckConstraint instance.

See also: TCheckConstraint (267)

### 10.15.2 Method overview

Page	Property	Description
270	Add	Add new TCheckConstraint item to the collection
269	Create	Create a new instance of the TCheckConstraints class.

### 10.15.3 Property overview

Page	Property	Access	Description
270	Items	rw	Indexed access to the items in the collection

### 10.15.4 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 (285) 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 (267) class pointer.

See also: TCheckConstraint (267), TDataset (285)

### 10.15.5 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 ([267](#)) instance to the collection. it returns the newly added instance.

**See also:** TCheckConstraint ([267](#)), #rtl.classes.TCollection.Add ([??](#))

### 10.15.6 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 ([??](#))

## 10.16 TCurrencyField

### 10.16.1 Description

TCurrencyField is the field class used by TDataset ([285](#)) when it needs to manage currency-valued data.(TField.Datatype ([346](#)) equals ftCurrency). It simply sets some Tfield ([334](#)) 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 ([334](#)), TDataset ([285](#))

### 10.16.2 Method overview

Page	Property	Description
<a href="#">271</a>	Create	Create a new instance of a TCurrencyField.

### 10.16.3 Property overview

Page	Property	Access	Description
<a href="#">271</a>	Currency		Is the field a currency field

### 10.16.4 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 ([271](#))) to be able to work with currency data.

**See also:** TField ([334](#)), TCurrencyField.Currency ([271](#))

### 10.16.5 TCurrencyField.Currency

**Synopsis:** Is the field a currency field

**Declaration:** Property Currency :

**Visibility:** published

**Access:**

**Description:** Currency is inherited from TFloatField.Currency ([372](#)) 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 ([372](#))

## 10.17 TCustomConnection

### 10.17.1 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 ([272](#))) and to deactivate the connection (TCustomConnection.Close ([272](#))), plus a property to inspect the state (Connected ([273](#))) of the connected.

**See also:** TCustomConnection.Open ([272](#)), TCustomConnection.Close ([272](#)), TCustomConnection.Connected ([273](#))

### 10.17.2 Method overview

Page	Property	Description
<a href="#">272</a>	Close	Close the connection
<a href="#">272</a>	Destroy	Remove the TCustomconnection instance from memory
<a href="#">272</a>	Open	Makes the connection to the server

### 10.17.3 Property overview

Page	Property	Access	Description
<a href="#">274</a>	AfterConnect	rw	Event triggered after a connection is made.
<a href="#">274</a>	AfterDisconnect	rw	Event triggered after a connection is closed
<a href="#">275</a>	BeforeConnect	rw	Event triggered before a connection is made.
<a href="#">275</a>	BeforeDisconnect	rw	Event triggered before a connection is closed
<a href="#">273</a>	Connected	rw	Is the connection established or not
<a href="#">273</a>	DataSetCount	r	Number of datasets connected to this connection
<a href="#">273</a>	DataSets	r	Datasets linked to this connection
<a href="#">274</a>	LoginPrompt	rw	Should the OnLogin be triggered
<a href="#">275</a>	OnLogin	rw	Event triggered when a login prompt is shown.

### 10.17.4 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` ([275](#)) 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` ([274](#)) event is triggered.

Calling the `Close` method is equivalent to setting the `Connected` ([273](#)) 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` ([248](#)) exception will be raised.

See also: `TCustomConnection.BeforeDisconnect` ([275](#)), `TCustomConnection.AfterDisconnect` ([274](#)), `TCustomConnection.Open` ([272](#)), `TCustomConnection.Connected` ([273](#))

### 10.17.5 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` ([272](#))

### 10.17.6 TCustomConnection.Open

Synopsis: Makes the connection to the server

Declaration: `procedure Open`

Visibility: public

Description: Open establishes the connection with the server if it was not yet connected. Calling this method first triggers the BeforeConnect ([275](#)) event. If an exception is raised during the execution of that event handler, the connect process is aborted. If LoginPrompt ([274](#)) is True, the OnLogin ([275](#)) event handler is called. Only after this event, the connection is actually established. After the connection was established, the AfterConnect ([274](#)) event is triggered.

Calling the Open method is equivalent to setting the Connected ([273](#)) 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 ([275](#)), TCustomConnection.LoginPrompt ([274](#)), TCustomConnection.OnLogin ([275](#)), TCustomConnection.AfterConnect ([274](#)), TCustomConnection.Connected ([273](#))

### **10.17.7 TCustomConnection.DataSetCount**

Synopsis: Number of datasets connected to this connection

Declaration: Property DataSetCount : LongInt

Visibility: public

Access: Read

Description: DataSetCount is the number of datasets connected to this connection component. The actual datasets are available through the Datasets ([273](#)) array property. As implemented in TCustomConnection, this property is always zero. Descendent classes implement the actual count.

See also: TDataset ([285](#)), TCustomConnection.Datasets ([273](#))

### **10.17.8 TCustomConnection.Datasets**

Synopsis: Datasets linked to this connection

Declaration: Property Datasets[Index: LongInt]: TDataset

Visibility: public

Access: Read

Description: Datasets allows indexed access to the datasets connected to this connection. Index is a zero-based indexed, it's maximum value is DatasetCount-1 ([273](#)).

See also: DatasetCount ([273](#))

### **10.17.9 TCustomConnection.Connected**

Synopsis: Is the connection established or not

Declaration: Property Connected : Boolean

Visibility: published

Access: Read,Write

**Description:** Connected is True if the connection to the server is established, False if it is disconnected.

The property can be set to True to establish a connection (equivalent to calling TCustomConnection.Open (272), or to False to break it (equivalent to calling TCustomConnection.Close (272).

See also: TCustomConnection.Open (272), TCustomConnection.Close (272)

### 10.17.10 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 (275)

### 10.17.11 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 (272). It can be used to open datasets, or indicate a connection status change.

See also: TCustomConnection.Open (272), TCustomConnection.BeforeConnect (275), TCustomConnection.OnLogin (275)

### 10.17.12 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 (272). It can be used for instance to indicate a connection status change.

See also: TCustomConnection.Close (272), TCustomConnection.BeforeDisconnect (275)

### 10.17.13 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 ([272](#)).

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 ([272](#)), TCustomConnection.AfterConnect ([274](#)), TCustomConnection.OnLogin ([275](#))

### 10.17.14 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 ([272](#)).

It can be used for instance to check for unsaved changes, to save these 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 ([272](#)), TCustomConnection.AfterDisconnect ([274](#))

### 10.17.15 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 ([274](#)) property is True, after the TCustomConnection.BeforeConnect ([275](#)) event, but before the connection is actually established.

**See also:** TCustomConnection.BeforeConnect ([275](#)), TCustomConnection.LoginPrompt ([274](#)), TCustomConnection.Open ([272](#))

## 10.18 TDatabase

### 10.18.1 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 (285), 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 (279) boolean property, which indicates whether a connection to the database is established or not. The actual logic to establish a connection to a database must be implemented by descendent classes.

See also: TDataset (285), TDatabase (276)

### 10.18.2 Method overview

Page	Property	Description
277	CloseDataSets	Close all connected datasets
277	CloseTransactions	End all transactions
276	Create	Initialize a new TDatabase class instance.
277	Destroy	Remove a TDatabase instance from memory.
278	EndTransaction	End an active transaction.
277	StartTransaction	Start a new transaction.

### 10.18.3 Property overview

Page	Property	Access	Description
279	Connected	rw	Is the database connected
279	DatabaseName	rw	Database name or path
278	Directory	rw	Directory for the database
279	IsSQLBased	r	Is the database SQL based.
279	KeepConnection	rw	Should the connection be kept active
280	Params	rw	Connection parameters
278	TransactionCount	r	Number of transaction components connected to this database.
278	Transactions	r	Indexed access to all transaction components connected to this database.

### 10.18.4 TDatabase.Create

**Synopsis:** Initialize a new TDatabase class instance.

**Declaration:** constructor Create(AOwner: TComponent); Override

**Visibility:** public

**Description:** Create initializes a new instance of the TDatabase class. It allocates some resources and then calls the inherited constructor.

See also: TDBDataset (328), TDBTransaction (329), TDatabase.Destroy (277)

### 10.18.5 **TDatabase.Destroy**

**Synopsis:** Remove a TDatabase instance from memory.

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

**Visibility:** public

**Description:** `Destroy` starts by disconnecting the database (thus closing all datasets and ending all transactions), then notifies all connected datasets and transactions that it is about to be released. After this, it releases all resources used by the TDatabase instance

**See also:** [TDatabase.CloseDatasets \(277\)](#)

### 10.18.6 **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 \(272\)](#), [TDatabase.CloseTransactions \(277\)](#)

### 10.18.7 **TDatabase.CloseTransactions**

**Synopsis:** End all transactions

**Declaration:** `procedure CloseTransactions`

**Visibility:** public

**Description:** `CloseTransaction` calls [TDBTransaction.EndTransaction \(329\)](#) on all connected transactions. It is called automatically when the connection is closed, after all datasets are closed.

**See also:** [TCustomConnection.Close \(272\)](#), [TDatabase.CloseDatasets \(277\)](#)

### 10.18.8 **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 shoud use a [TDBTransaction \(329\)](#) component instead and invoke the [TDBTransaction.StartTransaction \(329\)](#) method.

**See also:** [TDBTransaction \(329\)](#), [TDBTransaction.StartTransaction \(329\)](#)

### **10.18.9 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 shoud use a TDBTransaction ([329](#)) component instead and invoke the TDBTransaction.EndTransaction ([329](#)) method.

**See also:** TDBTransaction ([329](#)), TDBTransaction.EndTransaction ([329](#))

### **10.18.10 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 ([278](#)) array property.

**See also:** TDatabase.Transactions ([278](#))

### **10.18.11 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 ([278](#))

### **10.18.12 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 ([276](#)).

**See also:** TDatabase.Params ([280](#)), TDatabase.IsSQLBased ([279](#))

### **10.18.13 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 (280), TDatabase.Directory (278)

### **10.18.14 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 (273).

**See also:** TCustomConnection.Connected (273)

### **10.18.15 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 (278) 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 (278), TDatabase.Params (280)

### **10.18.16 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 (280)

### 10.18.17 TDatabase.Params

**Synopsis:** Connection parameters

**Declaration:** Property Params : TString

**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 (278), TDatabase.DatabaseName (279)

## 10.19 TDataLink

### 10.19.1 Description

TDataLink is used by GUI controls or datasets in a master-detail relationship to handle data events coming from a TDatasource (322) instance. It is a class that exists for component programmers, application coders should never need to use TDataLink or one of its descendants.

DB-Aware Component coders must use a TDatalink instance to handle all communication with a TDataset (285) instance, rather than communicating directly with the dataset. TDataLink contains methods which are called by the various events triggered by the dataset. Inversely, it has some methods to trigger actions in the dataset.

TDatalink is an abstract class; it is never used directly. Instead, a descendent class is used which overrides the various methods that are called in response to the events triggered by the dataset. Examples are .

See also: TDataset (285), TDatasource (322), TDetailDatalink (333), TMasterDataLink (388)

### 10.19.2 Method overview

Page	Property	Description
281	Create	Initialize a new instance of TDataLink
281	Destroy	Remove an instance of TDatalink from memory
281	Edit	Set the dataset in edit mode, if possible
282	ExecuteAction	Execute action
282	UpdateAction	Update handler for actions
282	UpdateRecord	Called when the data in the dataset must be updated

### 10.19.3 Property overview

Page	Property	Access	Description
282	Active	r	Is the link active
283	ActiveRecord	rw	Currently active record
283	BOF	r	Is the dataset at the first record
283	BufferCount	rw	Set to the number of record buffers this datalink needs.
284	DataSet	r	Dataset this datalink is connected to
284	DataSource	rw	Datasource this datalink is connected to
284	DataSourceFixed	rw	Can the datasource be changed
284	Editing	r	Is the dataset in edit mode
285	Eof	r	
285	ReadOnly	rw	Is the link readonly
285	RecordCount	r	Number of records in the buffer of the dataset

### 10.19.4 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 (281)

### 10.19.5 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 (281)

### 10.19.6 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 (323) method of the DataSource (284) to which the datalink instance is coupled. If the TDatasource.AutoEdit (324) 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 (322), TDatalink.DataSource (284), TDatasource.Edit (323), TDatasource.AutoEdit (324)

### **10.19.7 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 (280) notification (which could result from writing data to the dataset) will be blocked.

**See also:** TDatalink.RecordChange (280)

### **10.19.8 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 (282)

### **10.19.9 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 (282)

### **10.19.10 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 (314) whenever the DataSource (284) property is set.

See also: TDatalink.DataSource (284), TDatalink.ReadOnly (285), TDataset.Active (314)

### 10.19.11 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 (283), TDatalink.EOF (285)

### 10.19.12 TDataLink.BOF

**Synopsis:** Is the dataset at the first record

**Declaration:** Property BOF : Boolean

**Visibility:** public

**Access:** Read

**Description:** BOF returns TDataset.BOF (306) if the dataset is available, True otherwise.

See also: TDatalink.EOF (285), TDataset.BOF (306)

### 10.19.13 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 (290), TDatalink.ActiveRecord (283)

### **10.19.14 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 (284), TDataset (285)

### **10.19.15 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 (322) instance to get access to the dataset it is connected to. A datalink never points directly to a TDataset (285) 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 (285), TDatasource (322)

### **10.19.16 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 (284) 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 (284)

### **10.19.17 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 (281) 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 (281), TDataset.Edit (295)

### 10.19.18 TDataLink.Eof

Synopsis:

Declaration: Property Eof : Boolean

Visibility: public

Access: Read

Description: EOF returns TDataset.EOF ([308](#)) if the dataset is available, True otherwise.

See also: TDatalink.BOF ([283](#)), TDataset.EOF ([308](#))

### 10.19.19 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 ([281](#)) 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 ([282](#)) property can be used to disable delivery of events to the datalink.

See also: TDatalink.Active ([282](#)), TDatalink.edit ([281](#))

### 10.19.20 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 ([283](#)) property: RecordCount is always less than Buffercount.

See also: TDatalink.BufferCount ([283](#))

## 10.20 TDataSet

### 10.20.1 Description

TDataSet is the main class of the db unit. This abstract class provides all basic functionality to access data stored in tabular format: The data consists of records, and the data in each record is organised in several fields.

TDataSet has a buffer to cache a few records in memory, this buffer is used by TDatasource to create the ability to use data-aware components.

TDataSet is an abstract class, which provides the basic functionality to access, navigate through the data and - in case read-write access is available, edit existing or add new records.

TDataSet is an abstract class: it does not have the knowledge to store or load the records from whatever medium the records are stored on. Descendants add the functionality to load and save the data. Therefor TDataSet is never used directly, one always instantiates a descendent class.

Initially, no data is available: the dataset is inactive. The Open (303) 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 (314) property). Likewise, the Close (293) 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 (302), Prev (285), First (298) and Last (301) methods, it is possible to navigate through the records. The EOF (308) property will be True if the last row has been reached. Likewise, the BOF (306) 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 descendants that support counting the number of records, the RecordCount (311) will be zero.

The Append (291) and Insert (300) methods can be used to insert new records to the set of records. The TDataSet.Delete (294) statement is used to delete the current record, and the Edit (295) 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 (303) method, or can be undone using the Cancel (292) method.

The data in the various fields properties is available through the Fields (312) 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 (334)



### 10.20.2 Method overview

Page	Property	Description
290	ActiveBuffer	Currently active memory buffer
291	Append	Append a new record to the data
291	AppendRecord	Append a new record to the dataset and fill with data
292	BookmarkValid	Test whether ABookMark is a valid bookmark.
292	Cancel	Cancel the current editing operation
292	CheckBrowseMode	Check whether the dataset is in browse mode.
292	ClearFields	Clear the values of all fields
293	Close	Close the dataset
293	CompareBookmarks	Compare two bookmarks
293	ControlsDisabled	Check whether the controls are disabled
290	Create	Create a new TDataSet instance
294	CreateBlobStream	Create blob stream
294	CursorPosChanged	Indicate a change in cursor position
294	DataConvert	Convert data from/to native format
294	Delete	Delete the current record.
290	Destroy	Free a TDataSet instance
295	DisableControls	Disable event propagation of controls
295	Edit	Set the dataset in editing mode.
296	EnableControls	Enable event propagation of controls
296	FieldByName	Search a field by name
296	FindField	Find a field by name
297	FindFirst	Find the first active record (deprecated)
297	FindLast	Find the last active record (deprecated)
297	FindNext	Find the next active record (deprecated)
297	FindPrior	Find the previous active record (deprecated)
298	First	Position the dataset on the first record.
298	FreeBookmark	Free a bookmark obtained with GetBookmark (deprecated)
298	GetBookmark	Get a bookmark pointer (deprecated)
299	GetCurrentRecord	Copy the data for the current record in a memory buffer
290	GetFieldData	Get the data for a field
299	GetFieldList	Return field instances in a list
299	GetFieldNames	Return a list of all available field names
299	GotoBookmark	Jump to bookmark
300	Insert	Insert a new record at the current position.
300	InsertRecord	Insert a new record with given values.
300	IsEmpty	Check if the dataset contains no data
300	IsLinkedTo	Check whether a datasource is linked to the dataset
301	IsSequenced	Is the data sequenced
301	Last	Navigate forward to the last record
301	Locate	Locate a record based on some key values
302	Lookup	Search for a record and return matching values.
302	MoveBy	Move the cursor position
302	Next	Go to the next record in the dataset.
303	Open	Activate the dataset: Fetch data into memory.
303	Post	Post pending edits to the database.
304	Prior	Go to the previous record
304	Refresh	Refresh the records in the dataset
304	Resync	Resynchronize the data buffer
291	SetFieldData	Store the data for a field
304	SetFields	Set a number of field values at once
305	Translate	Transliterate a buffer
305	UpdateCursorPos	Update cursor position
305	UpdateRecord	Indicate that the record contents have changed
306	UpdateStatus	Get the update status for the current record

### 10.20.3 Property overview

Page	Property	Access	Description
314	Active	rw	Is the dataset open or closed.
318	AfterCancel	rw	Event triggered after a Cancel operation.
315	AfterClose	rw	Event triggered after the dataset is closed
318	AfterDelete	rw	Event triggered after a succesful Delete operation.
316	AfterEdit	rw	Event triggered after the dataset is put in edit mode.
316	AfterInsert	rw	Event triggered after the dataset is put in insert mode.
315	AfterOpen	rw	Event triggered after the dataset is opened.
317	AfterPost	rw	Event called after changes have been posted to the underlying database
319	AfterRefresh	rw	Event triggered after the data has been refreshed.
319	AfterScroll	rw	Event triggered after the cursor has changed position.
314	AutoCalcFields	rw	How often should the value of calculated fields be calculated
317	BeforeCancel	rw	Event triggered before a Cancel operation.
315	BeforeClose	rw	Event triggered before the dataset is closed.
318	BeforeDelete	rw	Event triggered before a Delete operation.
316	BeforeEdit	rw	Event triggered before the dataset is put in edit mode.
315	BeforeInsert	rw	Event triggered before the dataset is put in insert mode.
314	BeforeOpen	rw	Event triggered before the dataset is opened.
317	BeforePost	rw	Event called before changes are posted to the underlying database
319	BeforeRefresh	rw	Event triggered before the data is refreshed.
318	BeforeScroll	rw	Event triggered before the cursor changes position.
306	BlockReadSize	rw	Number of records to read
306	BOF	r	Is the cursor at the beginning of the data (on the first record)
306	Bookmark	rw	Get or set the current cursor position
307	CanModify	r	Can the data in the dataset be modified
308	DataSource	r	Datasource this dataset is connected to.
308	DefaultFields	r	Is the dataset using persisten fields or not.
308	EOF	r	Indicates whether the last record has been reached.
309	FieldCount	r	Number of fields
309	FieldDefs	rw	Definitions of available fields in the underlying database
312	Fields	r	Indexed access to the fields of the dataset.
312	FieldValues	rw	Acces to field values based on the field names.
313	Filter	rw	Filter to apply to the data in memory.
313	Filtered	rw	Is the filter active or not.
313	FilterOptions	rw	Options to apply when filtering
310	Found	r	Check success of one of the Find methods
310	IsUniDirectional	r	Is the dataset unidirectional (i.e. forward scrolling only)
310	Modified	r	Was the current record modified ?
320	OnCalcFields	rw	Event triggered when values for calculated fields must be computed.
320	OnDeleteError	rw	Event triggered when a delete operation fails.
321	OnEditError	rw	Event triggered when an edit operation fails.
321	OnFilterRecord	rw	Event triggered to filter records.
321	OnNewRecord	rw	Event triggered when a new record is created.
322	OnPostError	rw	Event triggered when a post operation fails.
311	RecNo	rw	Current record number
311	RecordCount	r	Number of records in the dataset
311	RecordSize	r	Size of the record in memory
312	State	r	Current operational state of the dataset

#### 10.20.4 TDataSet.Create

**Synopsis:** Create a new TDataSet instance

**Declaration:** constructor Create(AOwner: TComponent); Override

**Visibility:** public

**Description:** Create initializes a new TDataSet (285) 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 (290)

#### 10.20.5 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 (293), TDataSet.Create (290)

#### 10.20.6 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.

#### 10.20.7 TDataSet.GetFieldData

**Synopsis:** Get the data for a field

```
Declaration: function GetFieldData(Field: TField; Buffer: Pointer) : Boolean; Virtual
              ; Overload
              function GetFieldData(Field: TField; Buffer: Pointer;
                                   NativeFormat: Boolean) : Boolean; Virtual
              ; Overload
```

**Visibility:** public

**Description:** GetFieldData should copy the data for field Field from the internal dataset memory buffer into the memory pointed to by Buffer. This function is not intended for use by end-user applications, and should be used only in descendent classes, where it can be overridden. The function should return True if data was available and has been copied, or False if no data was available (in which case the field has value Null). The NativeFormat determines whether the data should be in native format (e.g. whether the date/time values should be in TDateTime format).

**Errors:** No checks are performed on the validity of the memory buffer

See also: [TField.DisplayText \(346\)](#)

### **10.20.8 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 \(346\)](#)

### **10.20.9 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 (300)` 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 \(300\)](#), [TDataset.Edit \(295\)](#)

### **10.20.10 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 (304)`) and attempts to post the record using `TDataset.Post (303)`. 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 \(291\)](#), [TDataset.SetFields \(304\)](#), [TDataset.Post \(303\)](#)

### **10.20.11 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 \(298\)](#), [TDataSet.SetBookmark \(285\)](#), [TDataSet.FreeBookmark \(298\)](#), [TDataSet.BookmarkAvailable \(285\)](#)

### **10.20.12 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.OnBeforeCancel (285) and TDataSet.OnAfterCancel (285) events. If the dataset was in insert mode, then the TDataSet.OnBeforeScroll (285) and TDataSet.OnAfterScroll (285) events are triggered after and respectively before the OnBeforeCancel and OnAfterCancel events.

If the dataset was not in one of the editing modes when Cancel is called, then nothing will happen.

**See also:** [TDataSet.State \(312\)](#), [TDataSet.Append \(291\)](#), [TDataSet.Insert \(300\)](#), [TDataSet.Edit \(295\)](#)

### **10.20.13 TDataSet.CheckBrowseMode**

**Synopsis:** Check whether the dataset is in browse mode.

**Declaration:** procedure CheckBrowseMode

**Visibility:** public

**Description:** CheckBrowseMode checks whether the dataset is in browse mode (State=dsBrowse). If it is not, an EDatabaseError (248) exception is raised.

**See also:** [TDataSet.State \(312\)](#)

### **10.20.14 TDataSet.ClearFields**

**Synopsis:** Clear the values of all fields

**Declaration:** procedure ClearFields

**Visibility:** public

**Description:** ClearFields clears the values of all fields.

**Errors:** If the dataset is not in editing mode (`State` in `dsEditmodes`), then an `EDatabaseError` (248) exception will be raised.

See also: `TDataSet.State` (312), `TField.Clear` (338)

### 10.20.15 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.OnBeforeClose` (285) and `TDataSet.OnAfterClose` (285) events. If the dataset is not active, nothing happens.

**Errors:** If an exception occurs during the closing of the dataset, the `OnAfterClose` event will not be triggered.

See also: `TDataSet.Active` (314), `TDataSet.Open` (303)

### 10.20.16 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` (295) call can be used to disable sending of data events to the controls. The sending can be re-enabled with `TDataSet.EnableControls` (296). 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` (295), `TDataSet.EnableControls` (296)

### 10.20.17 TDataSet.CompareBookmarks

**Synopsis:** Compare two bookmarks

**Declaration:** `function CompareBookmarks(Bookmark1: TBookmark; Bookmark2: TBookmark) : LongInt; Virtual`

**Visibility:** public

**Description:** `CompareBookmarks` can be used to compare the relative positions of 2 bookmarks. It returns a negative value if `Bookmark1` is located before `Bookmark2`, zero if they refer to the same record, and a positive value if the second bookmark appears before the first bookmark. This function must be overridden by descendent classes of `TDataSet`. The implementation in `TDataSet` always returns zero.

**Errors:** No checks are performed on the validity of the bookmarks.

See also: `TDataSet.BookmarkValid` (292), `TDataSet.GetBookmark` (298), `TDataSet.SetBookmark` (285)

### 10.20.18 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 (262) and TBlobField.SaveToStream (263) 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 (262), TBlobField.SaveToStream (263)

### 10.20.19 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).

### 10.20.20 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 (290), TDataset.SetFieldData (291)

### 10.20.21 TDataSet.Delete

**Synopsis:** Delete the current record.

**Declaration:** procedure Delete

Visibility: public

Description: Delete will delete the current record. This action will trigger the TDataset.BeforeDelete (318), TDataset.BeforeScroll (318), TDataset.AfterDelete (318) and TDataset.AfterScroll (319) 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 (248) exception will be raised.

See also: TDataset.Cancel (292), TDataset.BeforeDelete (318), TDataset.BeforeScroll (318), TDataset.AfterDelete (318), TDataset.AfterScroll (319)

## 10.20.22 TDataset.DisableControls

Synopsis: Disable event propagation of controls

Declaration: procedure DisableControls

Visibility: public

Description: DisableControls tells the dataset to stop sending data-related events to the controls. This can be used before starting operations that will cause the current record to change a lot, or before any other lengthy operation that may cause a lot of events to be sent to the controls that show data from the dataset: each event will cause the control to update itself, which is a time-consuming operation that may also cause a lot of flicker on the screen.

The sending of events to the controls can be re-enabled with Tdataset.EnableControls (296). 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 (293).

See also: TDataset.EnableControls (296), TDataset.ControlsDisabled (293)

## 10.20.23 TDataset.Edit

Synopsis: Set the dataset in editing mode.

Declaration: procedure Edit

Visibility: public

Description: Edit will set the dataset in edit mode: the contents of the current record can then be changed. This action will call the TDataset.BeforeEdit (316) and TDataset.AfterEdit (316) 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 (291) instead.

Errors: If the dataset is read-only or not opened, then an EDatabaseError (248) exception will be raised.

See also: TDataset.State (312), TDataset.EOF (308), TDataset.BOF (306), TDataset.Append (291), TDataset.BeforeEdit (316), TDataset.AfterEdit (316)

### 10.20.24 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` (295) 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` (293).

**See also:** `TDataSet.DisableControls` (295), `TDataSet.ControlsDisabled` (293)

### 10.20.25 TDataSet.FieldName

**Synopsis:** Search a field by name

**Declaration:** function FieldByName(const FieldName: string) : TField

**Visibility:** public

**Description:** `FieldByName` is a shortcut for `Fields.FieldByName` (367): it searches for the field with `fieldname` equalling `FieldName`. The case is performed case-insensitive. The matching field instance is returned.

**Errors:** If the field is not found, an `EDatabaseError` (248) exception will be raised.

**See also:** `TFIELDS.FieldByname` (367), `TDataSet.FindField` (296)

### 10.20.26 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` (367): it searches for the field with `fieldname` equalling `FieldName`. The case is performed case-insensitive. The matching field instance is returned, and if no match is found, `Nil` is returned.

**See also:** `TDataSet.FieldByname` (296), `TFIELDS.FindField` (367)

### **10.20.27 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 descendants of TDataSet: The implementation in TDataSet always returns False, indicating that the position was not changed.

This method is deprecated, use TDataSet.First ([298](#)) instead.

**See also:** TDataSet.First ([298](#)), TDataSet.FindLast ([297](#)), TDataSet.FindNext ([297](#)), TDataSet.FindPrior ([297](#))

### **10.20.28 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 descendants of TDataSet: The implementation in TDataSet always returns False, indicating that the position was not changed.

This method is deprecated, use TDataSet.Last ([301](#)) instead.

**See also:** TDataSet.Last ([301](#)), TDataSet.FindFirst ([297](#)), TDataSet.FindNext ([297](#)), TDataSet.FindPrior ([297](#))

### **10.20.29 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 descendants of TDataSet: The implementation in TDataSet always returns False, indicating that the position was not changed.

This method is deprecated, use TDataSet.Next ([302](#)) instead.

**See also:** TDataSet.Next ([302](#)), TDataSet.FindFirst ([297](#)), TDataSet.FindLast ([297](#)), TDataSet.FindPrior ([297](#))

### **10.20.30 TDataSet.FindPrior**

**Synopsis:** Find the previous active record (deprecated)

**Declaration:** function FindPrior : Boolean; Virtual

**Visibility:** public

**Description:** `FindPrior` positions the cursor on the previous record (taking into account filtering), and returns `True` if the cursor position was changed. This method must be implemented by descendants of `TDataset`: The implementation in `TDataset` always returns `False`, indicating that the position was not changed.

This method is deprecated, use `TDataset.Prior` (304) instead.

See also: `TDataset.Prior` (304), `TDataset.FindFirst` (297), `TDataset.FindLast` (297), `TDataset.FindPrior` (297)

### 10.20.31 TDataSet.First

**Synopsis:** Position the dataset on the first record.

**Declaration:** `procedure First`

**Visibility:** `public`

**Description:** `First` positions the dataset on the first record. This action will trigger the `TDataset.BeforeScroll` (318) and `TDataset.AfterScroll` (319) events. After the action is completed, the `TDataset.BOF` (306) property will be `True`.

**Errors:** If the dataset is unidirectional or is closed, an `EDatabaseError` (248) exception will be raised.

See also: `TDataset.Prior` (304), `TDataset.Last` (301), `TDataset.Next` (302), `TDataset.BOF` (306), `TDataset.BeforeScroll` (318), `TDataset.AfterScroll` (319)

### 10.20.32 TDataSet.FreeBookmark

**Synopsis:** Free a bookmark obtained with `GetBookmark` (deprecated)

**Declaration:** `procedure FreeBookmark (ABookmark: TBookmark); Virtual`

**Visibility:** `public`

**Description:** `FreeBookmark` must be used to free a bookmark obtained by `TDataset.GetBookmark` (298). It should not be used on bookmarks obtained with the `TDataset.Bookmark` (306) 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` (298), `TDataset.Bookmark` (306)

### 10.20.33 TDataSet.GetBookmark

**Synopsis:** Get a bookmark pointer (deprecated)

**Declaration:** `function GetBookmark : TBookmark; Virtual`

**Visibility:** `public`

**Description:** `GetBookmark` gets a bookmark pointer to the current cursor location. The `TDataset.SetBookmark` (285) 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` (298) call. The bookmark will be `Nil` if the dataset is empty or not active.

This call is deprecated. Use the `TDataset.Bookmark` (306) property instead to get a bookmark.

See also: `TDataset.SetBookmark` (285), `TDataset.FreeBookmark` (298), `TDataset.Bookmark` (306)

### 10.20.34 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 descendants 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 ([290](#))

### 10.20.35 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 ([248](#)) exception will be raised.

**See also:** TDataset.GetFieldNames ([299](#)), TDataset.FieldName ([296](#)), TDataset.FindField ([296](#))

### 10.20.36 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 ([285](#)), TDataset.FieldName ([296](#)), TDataset.FindField ([296](#))

### 10.20.37 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 ([298](#)) function.

This function is deprecated, use the TDataset.Bookmark ([306](#)) property instead.

**Errors:** if ABookmark does not contain a valid bookmark, then an exception may be raised.

**See also:** TDataset.Bookmark ([306](#)), TDataset.GetBookmark ([298](#)), TDataset.FreeBookmark ([298](#))

### 10.20.38 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 (315), BeforeScroll (318), OnNewRecord (321), AfterInsert (316) and AfterScroll (319) 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 (303) must be called.

**Errors:** If the dataset is read-only, calling Insert will result in an EDatabaseError (248).

**See also:** BeforeInsert (315), BeforeScroll (318), OnNewRecord (321), AfterInsert (316), AfterScroll (319), TDataset.Post (303), TDataset.Append (291)

### 10.20.39 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 (300), TDataset.SetFieldValues (285)

### 10.20.40 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 (308) and TDataset.BOF (306) are both True, and the dataset is not in insert mode.

**See also:** TDataset.EOF (308), TDataset.BOF (306), TDataset.State (312)

### 10.20.41 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" (324) points to the current dataset instance, or indirectly.

**See also:** TDatasource.Dataset (324)

### 10.20.42 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` (311) property to navigate in the records of the data. By default, this property is set to `True`, but `TDataSet` descendants 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` (311)

### 10.20.43 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` (302). After a call to `Last`, the `TDataSet.EOF` (308) property will be `True`.

Calling this method will trigger the `TDataSet.BeforeScroll` (318) and `TDataSet.AfterScroll` (319) events.

**See also:** `TDataSet.First` (298), `TDataSet.Next` (302), `TDataSet.EOF` (308), `TDataSet.BeforeScroll` (318), `TDataSet.AfterScroll` (319)

### 10.20.44 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` (241).

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` descendants to implement this method and return an appropriate value.

**See also:** `TDataSet.Find` (285), `TDataSet.Lookup` (302), `TLocateOption` (241)

### 10.20.45 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 False in TDataSet. Descendents of TDataSet can override this method to call TDataSet.Locate (301) 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 (248) exception will be raised.

**See also:** TDataSet.Locate (301)

### 10.20.46 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 (318) and TDataSet.AfterScroll (319) 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 (248) exception on unidirectional datasets.

**See also:** TDataSet.RecNo (311), TDataSet.BeforeScroll (318), TDataSet.AfterScroll (319)

### 10.20.47 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 (318) and TDataSet.AfterScroll (319) events. If the dataset is located on the last known record (EOF (308) 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 (248) exception.

**See also:** TDataSet.MoveBy (302), TDataSet.Prior (304), TDataSet.Last (301), TDataSet.BeforeScroll (318), TDataSet.AfterScroll (319), TDataSet.EOF (308)

### 10.20.48 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 initialises the TDataSet and brings the dataset in a browsable state:

Effectively the following happens:

- 1.The BeforeOpen event is triggered.
- 2.The descendants InternalOpen method is called to actually fetch data and initialize field-defs and field instances.
- 3.BOF ([306](#))is set to True
- 4.Internal buffers are allocated and filled with data
- 5.If the dataset is empty, EOF ([308](#)) is set to true
- 6.State ([312](#)) is set to dsBrowse
- 7.The AfterOpen ([315](#)) 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 ([314](#)), TDataSet.State ([312](#)), TDataSet.BOF ([306](#)), TDataSet.EOF ([308](#)), TDataSet.BeforeOpen ([314](#)), TDataSet.AfterOpen ([315](#))

### 10.20.49 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 ([300](#)), Append ([291](#)) or TDataSet.Edit ([295](#)) operation. The changes will be committed to memory - and usually immediatly 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 ([317](#)), AfterPost ([317](#)) events. After the call to Post, the State ([312](#)) 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 ([232](#))) will result in an EdatabaseError ([248](#)) exception. If an exception occurs during the save operation, the OnPostError ([322](#)) event is triggered to handle the error.

**See also:** Insert ([300](#)), Append ([291](#)), Edit ([295](#)), OnPostError ([322](#)), BeforePost ([317](#)), AfterPost ([317](#)), State ([312](#))

### 10.20.50 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 (318) and TDataSet.AfterScroll (319) events. If the dataset is located on the first record, (BOF (306) 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 (248) exception.

**See also:** TDataSet.MoveBy (302), TDataSet.Next (302), TDataSet.First (298), TDataSet.BeforeScroll (318), TDataSet.AfterScroll (319), TDataSet.BOF (306)

### 10.20.51 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 (319) and AfterRefresh (319) events.

**Errors:** Refreshing may fail if the underlying dataset descendent does not support it.

**See also:** TDataSet.Close (293), TDataSet.Open (303), BeforeRefresh (319), AfterRefresh (319)

### 10.20.52 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 (304) should be used. The Resync parameter indicates how the buffers should be refreshed.

**See also:** TDataSet.Refresh (304)

### 10.20.53 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 (312) property, and works its way down the array.

**Errors:** If the dataset is not in edit mode, then an EDatabaseError (248) exception will be raised. If there are more values than fields, an EListError exception will be raised.

See also: TDataset.Fields (312)

#### 10.20.54 TDataset.Translate

**Synopsis:** Transliterate a buffer

**Declaration:** function Translate(Src: PChar; Dest: PChar; ToOem: Boolean) : Integer  
; Virtual

**Visibility:** public

**Description:** Translate is called for all string fields for which the TStringField.Transliterate (414) 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 (414)

#### 10.20.55 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 (304)

#### 10.20.56 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 (285)

### **10.20.57 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.

### **10.20.58 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 ([231](#)), DisableControls ([231](#))

### **10.20.59 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 ([298](#)). As soon as TDataSet.Next ([302](#)) is called, BOF will no longer be true.

**See also:** TDataSet.EOF ([308](#)), TDataSet.Next ([302](#)), TDataSet.First ([298](#))

### **10.20.60 TDataSet.Bookmark**

**Synopsis:** Get or set the current cursor position

**Declaration:** Property Bookmark : TBookmarkStr

**Visibility:** public

**Access:** Read,Write

**Description:** Bookmark can be read to obtain a bookmark to the current position in the dataset. The obtained value can be used to return to current position at a later stage. Writing the Bookmark property with a value previously obtained like this, will reposition the dataset on the same position as it was when the property was read.

This is often used when scanning all records, like this:

```
Var
  B : 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 (295) and TDataset.EnableControls (296) 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 (295), TDataset.EnableControls (296)

### 10.20.61 TDataset.CanModify

**Synopsis:** Can the data in the dataset be modified

**Declaration:** 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 (300), TDataset.Append (291), TDataset.Delete (294), Tdataset.Edit (295)

### **10.20.62 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 ([322](#))

### **10.20.63 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 ([309](#)) 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 ([309](#)), TDataSet.Fields ([312](#)), TField ([334](#))

### **10.20.64 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 ([302](#)) method is called.
- 2.The TDataSet.Last ([301](#)) method is called (which is equivalent to moving to the last record and calling TDataSet.Next ([302](#))).
- 3.The dataset is empty when opened.

In all other cases, EOF is False. Note: when the cursor is on the last-but-one record, and Next is called (moving the cursor to the last record), EOF will not yet be True. Only if both the cursor is on the last record and Next is called, will EOF become True.

This means that the following loop will stop after the last record was visited:

```
With MyDataset do
  While not EOF do
    begin
      DoSomething;
      Next;
    end;
```

See also: TDataset.BOF (306), TDataset.Next (302), TDataset.Last (301), TDataset.IsEmpty (300)

### 10.20.65 TDataset.FieldCount

**Synopsis:** Number of fields

**Declaration:** Property FieldCount : LongInt

**Visibility:** public

**Access:** Read

**Description:** FieldCount is the same as Fields.Count (369), i.e. the number of fields. For a dataset with persistent fields (when DefaultFields (308) 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 (365)

### 10.20.66 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 (308) is True), then for each item in this list, a field 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 (303), TDataset.DefaultFields (308), TDataset.Fields (312)

### 10.20.67 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 (297), TDataset.FindLast (297), TDataset.FindNext (297) or TDataset.FindPrior (297) operations was successful.

**See also:** TDataset.FindFirst (297), TDataset.FindLast (297), TDataset.FindNext (297), TDataset.FindPrior (297)

### 10.20.68 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 (295) or Tdataset.Insert (300). It becomes True if a value was written to one of the fields of the dataset.

**See also:** Tdataset.Edit (295), TDataset.Insert (300), TDataset.Append (291), TDataset.Cancel (292), TDataset.Post (303)

### 10.20.69 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 (304) or TDataset.Next (302)), random positioning of the cursor, editing or filtering will result in an EDatabaseError (248). 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 (304), TDataset.Next (302)

### 10.20.70 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 ([311](#))

### 10.20.71 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 descendants, 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 ([311](#))

### 10.20.72 TDataSet.RecordSize

**Synopsis:** Size of the record in memory

**Declaration:** Property RecordSize : Word

**Visibility:** public

**Access:** Read

**Description:** RecordSize is the total size of the memory buffer used for the records. This property returns always 0 in the TDataSet implementation. Descendent classes should implement this property. Note that this property does not necessarily reflect the actual data size for the records. that may be more or less, depending on how the TDataSet descendent manages it's data.

See also: TField.Datasize ([346](#)), TDataset.RecordCount ([311](#)), TDataset.RecNo ([311](#))

### **10.20.73 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 ([314](#)), TDataset.Edit ([295](#)), TDataset.Insert ([300](#)), TDataset.Append ([291](#)), TDataset.Post ([303](#)), TDataset.Cancel ([292](#))

### **10.20.74 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 ([365](#)) 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 ([362](#)), TField ([334](#))

### **10.20.75 TDataSet.FieldValues**

**Synopsis:** Acces 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:

```
FieldName.FieldName).AsVariant
```

It can be read as well as written.

**See also:** TFields.FieldName ([367](#))

### 10.20.76 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 \(313\)](#), [TDataSet.FilterOptions \(313\)](#)

### 10.20.77 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 (313)` 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 \(313\)](#), [TDataSet.FilterOptions \(313\)](#)

### 10.20.78 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 (313)`, 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 \(313\)](#), [TDataSet.FilterOptions \(313\)](#)

### 10.20.79 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 (312) is then dsInactive). Setting the Active property to True is equivalent to calling TDataSet.Open (303), setting it to False is equivalent to calling TDataSet.Close (293)

**See also:** TDataSet.State (312), TDataSet.Open (303), TDataSet.Close (293)

### 10.20.80 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 (320) event.

**See also:** TField.FieldKind (353), TDataSet.OnCalcFields (320)

### 10.20.81 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 (315), TDataSet.State (312)

### **10.20.82 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` (319) 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` (315), `TDataSet.State` (312), `TDataSet.AfterScroll` (319)

### **10.20.83 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` (315), `TDataSet.State` (312)

### **10.20.84 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` (315), `TDataSet.State` (312)

### **10.20.85 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 ([291](#)) or TDataSet.Insert ([300](#)) 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 ([316](#)), TDataSet.Append ([291](#)), TDataSet.Insert ([300](#))

### 10.20.86 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 ([319](#)) 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 ([315](#)), TDataSet.AfterScroll ([319](#)), TDataSet.Append ([291](#)), TDataSet.Insert ([300](#))

### 10.20.87 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 ([295](#)) 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 ([316](#)), TDataSet.Edit ([295](#)), TDataSet.State ([312](#))

### 10.20.88 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 ([316](#)), TDataSet.Edit ([295](#)), TDataSet.State ([312](#))

### **10.20.89 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 ([303](#)) 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 ([303](#)), TDataset.AfterPost ([317](#)), TDataset.State ([312](#))

### **10.20.90 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 ([303](#)) operation was successfully completed, and the dataset is again in dsBrowse state. If an error occurred during the post operation, then the AfterPost event is not called, but the TDataset.OnPostError ([322](#)) event is triggered instead.

See also: TDataset.BeforePost ([317](#)), TDataset.Post ([303](#)), TDataset.State ([312](#)), TDataset.OnPostError ([322](#))

### **10.20.91 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 ([292](#)) 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 ([318](#)), TDataset.Cancel ([292](#)), TDataset.State ([312](#))

### 10.20.92 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 ([292](#)) operation was successfully completed, and the dataset is again in dsBrowse state.

**See also:** TDataset.BeforeCancel ([317](#)), TDataset.Cancel ([292](#)), TDataset.State ([312](#))

### 10.20.93 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 ([294](#)) 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 ([318](#)) event. If the dataset was in insert mode when the Delete method was called, then the event will not be called, as TDataset.Cancel ([292](#)) is called instead.

**See also:** TDataset.AfterDelete ([318](#)), TDataset.Delete ([294](#)), TDataset.BeforeScroll ([318](#)), TDataset.Cancel ([292](#)), TDataset.State ([312](#))

### 10.20.94 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 ([294](#)) operation, when the dataset is again in dsBrowse state. The event is followed by a TDataset.AfterScroll ([319](#)) event.

**See also:** TDataset.BeforeDelete ([318](#)), TDataset.Delete ([294](#)), TDataset.AfterScroll ([319](#)), TDataset.State ([312](#))

### 10.20.95 TDataSet.BeforeScroll

**Synopsis:** Event triggered before the cursor changes position.

**Declaration:** Property BeforeScroll : TDataSetNotifyEvent

**Visibility:** public

Access: Read,Write

Description: BeforeScroll is triggered before the cursor changes position. This can happen with one of the navigation methods: TDataSet.Next (302), TDataSet.Prior (304), TDataSet.First (298), TDataSet.Last (301), but also with two of the editing operations:TDataSet.Insert (300) and TDataSet.Delete (294). Raising an exception in this event handler aborts the operation in progress.

See also: TDataSet.AfterScroll (319), TDataSet.Next (302), TDataSet.Prior (304), TDataSet.First (298), TDataSet.Last (301), TDataSet.Insert (300), TDataSet.Delete (294)

### 10.20.96 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 (302), TDataSet.Prior (304), TDataSet.First (298), TDataSet.Last (301), but also with two of the editing operations:TDataSet.Insert (300) and TDataSet.Delete (294) 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 (319), TDataSet.Next (302), TDataSet.Prior (304), TDataSet.First (298), TDataSet.Last (301), TDataSet.Insert (300), TDataSet.Delete (294), TDataSet.Open (303)

### 10.20.97 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 (304) 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 (304), TDataSet.AfterRefresh (319), TDataSet.State (312)

### 10.20.98 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 ([304](#)) 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` ([304](#)), `TDataSet.State` ([312](#)), `TDataSet.BeforeRefresh` ([319](#))

### 10.20.99 **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` ([314](#)) 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` ([314](#)), `TField.Kind` ([334](#))

### 10.20.100 **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` ([294](#)) 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` ([235](#)) event handler type.

See also: `TDataSetErrorEvent` ([235](#)), `TDataSet.Delete` ([294](#)), `TDataSet.OnEditError` ([321](#)), `TDataSet.OnPostError` ([322](#))

### 10.20.101 TDataSet.OnEditError

**Synopsis:** Event triggered when an edit operation fails.

**Declaration:** Property OnEditError : TDataSetErrorEvent

**Visibility:** public

**Access:** Read,Write

**Description:** OnEditError is triggered when the TDataSet.Edit ([295](#)) 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 ([235](#)) event handler type.

**See also:** TDataSetErrorEvent ([235](#)), TDataSet.Edit ([295](#)), TDataSet.OnDeleteError ([320](#)), TDataSet.OnPostError ([322](#))

### 10.20.102 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 ([313](#)) 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 ([313](#)), TDataset.Filtered ([313](#)), TDataset.state ([312](#))

### 10.20.103 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 ([291](#)) or TDataSet.Insert ([300](#)) 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 ([312](#)) is again dsBrowse).

**See also:** TDataset.Append ([291](#)), TDataset.Insert ([300](#)), TDataset.State ([312](#))

### 10.20.104 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 ([303](#)) 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 ([235](#)) event handler type.

**See also:** TDataSetErrorEvent ([235](#)), TDataSet.Post ([303](#)), TDataSet.OnDeleteError ([320](#)), TDataSet.OnEditError ([321](#))

## 10.21 TDataSource

### 10.21.1 Description

TDatasource is a mediating component: it handles communication between any DB-Aware component (often edit controls on a form) and a TDataSet ([285](#)) instance. Any database aware component should never communicate with a dataset directly. Instead, it should communicate with a TDataSource ([322](#)) 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 ([285](#)), TDatalink ([280](#))

### 10.21.2 Method overview

Page	Property	Description
<a href="#">323</a>	Create	Create a new instance of TDataSource
<a href="#">323</a>	Destroy	Remove a TDataSource instance from memory
<a href="#">323</a>	Edit	Put the dataset in edit mode, if needed
<a href="#">324</a>	IsLinkedTo	Check if a dataset is linked to a certain dataset

### 10.21.3 Property overview

Page	Property	Access	Description
324	AutoEdit	rw	Should the dataset be put in edit mode automatically
324	DataSet	rw	Dataset this datasource is connected to
325	Enabled	rw	Enable or disable sending of events
325	OnDataChange	rw	Called whenever data changes in the current record
325	OnStateChange	rw	Called whenever the state of the dataset changes
326	OnUpdateData	rw	Called whenever the data in the dataset must be updated
324	State	r	State of the dataset

### 10.21.4 TDataSource.Create

**Synopsis:** Create a new instance of TDatasource

**Declaration:** constructor Create (AOwner: TComponent); Override

**Visibility:** public

**Description:** Create initializes a new instance of TDatasource. It simply allocates some resources and then calls the inherited constructor.

**See also:** TDatasource.Destroy (323)

### 10.21.5 TDataSource.Destroy

**Synopsis:** Remove a TDatasource instance from memory

**Declaration:** destructor Destroy; Override

**Visibility:** public

**Description:** Destroy notifies all TDataLink (280) 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 (323), TDatalink (280)

### 10.21.6 TDataSource.Edit

**Synopsis:** Put the dataset in edit mode, if needed

**Declaration:** procedure Edit

**Visibility:** public

**Description:** Edit will check AutoEdit (324): if it is True, then it puts the Dataset (324) 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 (295) directly.

**Errors:** An EDatabaseError (248) 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 (324), TDataset.Edit (295), TDataset.State (312)

### **10.21.7 TDataSource.IsLinkedTo**

**Synopsis:** Check if a dataset is linked to a certain dataset

**Declaration:** function IsLinkedTo(ADatasource: TDataSource) : Boolean

**Visibility:** public

**Description:** IsLinkedTo checks if it is somehow linked to ADatasource: it checks the Dataset ([324](#)) 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 ([324](#))

### **10.21.8 TDataSource.State**

**Synopsis:** State of the dataset

**Declaration:** Property State : TDataSetState

**Visibility:** public

**Access:** Read

**Description:** State contains the State ([312](#)) 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 ([312](#)) directly, to take into account the effect of the Enabled ([325](#)) property.

**See also:** TDataset.State ([312](#)), TDatasource.Enabled ([325](#))

### **10.21.9 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 ([323](#)) 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 ([323](#)), TDataset.Edit ([295](#))

### **10.21.10 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 applictaion programmer to the TDataset (285) 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 \(285\)](#), [TDataSource.Enabled \(325\)](#)

### 10.21.11 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 \(285\)](#), [TDataSource.Dataset \(324\)](#), [TDataSource.AutoEdit \(324\)](#)

### 10.21.12 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 (312) 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 \(326\)](#), [TDataSource.OnStateChange \(325\)](#), [TDataSet.State \(312\)](#), [TDataSource.Enabled \(325\)](#)

### 10.21.13 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 \(326\)](#), [TDataSource.OnStateChange \(325\)](#), [TDataSet.AfterScroll \(319\)](#), [TField.OnChange \(357\)](#), [TDataSource.Enabled \(325\)](#)

### 10.21.14 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 ([303](#)) 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 ([325](#)), TDatasource.OnStateChange ([325](#)), TDataSet.Post ([303](#))

## 10.22 TDateField

### 10.22.1 Description

TDateField is the class used when a dataset must manage data of type date. (TField.DataType ([346](#)) equals ftDate). It initializes some of the properties of the TField ([334](#)) 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 ([285](#)), TField ([334](#)), TDateTimeField ([326](#)), TTimeField ([415](#))

### 10.22.2 Method overview

Page	Property	Description
<a href="#">326</a>	Create	Create a new instance of a TDateField class.

### 10.22.3 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 ([334](#)) properties to configure the instance for working with date values.

See also: TField ([334](#))

## 10.23 TDateTimeField

### 10.23.1 Description

TDateTimeField is the class used when a dataset must manage data of type datetime. (TField.DataType ([346](#)) equals ftDateTime). It also serves as base class for the TDateField ([326](#)) or TTimeField ([415](#)) classes. It overrides some of the properties and methods of the TField ([334](#)) 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 \(285\)](#), [TField \(334\)](#), [TDateField \(326\)](#), [TTimeField \(415\)](#)

### 10.23.2 Method overview

Page	Property	Description
<a href="#">327</a>	Create	Create a new instance of a <code>TDateTimeField</code> class.

### 10.23.3 Property overview

Page	Property	Access	Description
<a href="#">327</a>	DisplayFormat	rw	Formatting string for textual representation of the field
<a href="#">328</a>	EditMask		Specify an edit mask for an edit control
<a href="#">327</a>	Value	rw	Contents of the field as a <code>TDateTime</code> value

### 10.23.4 `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` ([334](#)) properties to configure the instance for working with date/time values.

See also: [TField \(334\)](#)

### 10.23.5 `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` ([350](#)) by `TDateTimeField` as a `TDateTime` value. It returns the same value as the `TField.AsDateTime` ([342](#)) property.

See also: [TField.AsDateTime \(342\)](#), [TField.Value \(350\)](#)

### 10.23.6 `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` ([346](#)) property to format the value with the `DateTimeToString` ([??](#)) function.

See also: `DateTimeToString` ([??](#)), `FormatDateTime` ([??](#)), `TField.DisplayText` ([346](#))

### 10.23.7 TDateTimeField.EditMask

**Synopsis:** Specify an edit mask for an edit control

**Declaration:** Property `EditMask` :

**Visibility:** published

**Access:**

**Description:** `EditMask` can be used to specify an edit mask for controls that allow to edit this field. It has no effect on the field value, and serves only to ensure that the user can enter only correct data for this field.

`TDateTimeField` just changes the visibility of the `EditMark` property, it is introduced in `TField`.

For more information on valid edit masks, see the documentation of the GUI controls.

See also: `TField.EditMask` ([347](#))

## 10.24 TDBDataset

### 10.24.1 Description

`TDBDataset` is a `TDataset` descendent which introduces the concept of a database: a central component (`TDatabase` ([276](#))) which represents a connection to a database. This central component is exposed in the `TDBDataset.Database` ([329](#)) 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` ([329](#)) 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` ([276](#)), `TDBTransaction` ([329](#))

### 10.24.2 Method overview

Page	Property	Description
<a href="#">329</a>	<code>destroy</code>	Remove the <code>TDBDataset</code> instance from memory.

### 10.24.3 Property overview

Page	Property	Access	Description
<a href="#">329</a>	<code>DataBase</code>	<code>rw</code>	Database this dataset is connected to
<a href="#">329</a>	<code>Transaction</code>	<code>rw</code>	Transaction in which this dataset is running.

#### **10.24.4 TDBDataset.destroy**

**Synopsis:** Remove the TDBDataset instance from memory.

**Declaration:** destructor destroy;   Override

**Visibility:** public

**Description:** Destroy will disconnect the TDBDataset from its Database (329) and Transaction (329). After this it calls the inherited destructor.

**See also:** TDBDataset.Database (329), TDatabase (276)

#### **10.24.5 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 (276) 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 (248) exception will be raised.

**See also:** TDatabase (276), TDBDataset.Transaction (329)

#### **10.24.6 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 (329) 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 (329), TDBDataset.Database (329)

### **10.25 TDBTransaction**

#### **10.25.1 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 (276) instance, and to keep track of TDataSet instances which are connected to the transaction.

**See also:** TDatabase (276), TDataSet (285)

### 10.25.2 Method overview

Page	Property	Description
<a href="#">330</a>	CloseDataSets	Close all connected datasets
<a href="#">330</a>	Create	Transaction property
<a href="#">330</a>	destroy	Remove a TDBTransaction instance from memory.

### 10.25.3 Property overview

Page	Property	Access	Description
<a href="#">331</a>	Active	rw	Is the transaction active or not
<a href="#">331</a>	DataBase	rw	Database this transaction is connected to

### 10.25.4 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 ([330](#))

### 10.25.5 TDBTransaction.destroy

Synopsis: Remove a TDBTransaction instance from memory.

Declaration: destructor destroy; Override

Visibility: public

Description: Destroy first disconnects all connected TDBDataset ([328](#)) instances and then cleans up the resources allocated in the Create ([330](#)) constructor. After that it calls the inherited destructor.

See also: TDBTransaction.Create ([330](#))

### 10.25.6 TDBTransaction.CloseDataSets

Synopsis: Close all connected datasets

Declaration: procedure CloseDataSets

Visibility: public

Description: CloseDataSets closes all connected datasets (All TDBDataset ([328](#)) instances whose Transaction ([329](#)) property points to this TDBTransaction instance).

See also: TDBDataset ([328](#)), TDBDataset.Transaction ([329](#))

### 10.25.7 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 ([248](#)) exception.

See also: TDBTransaction.Active ([331](#)), TDatabase ([276](#))

### 10.25.8 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 ([329](#)). Reversely, setting Active to True will call StartTransaction, setting it to False will call TDBTransaction.EndTransaction ([329](#)).

See also: TDBTransaction.StartTransaction ([329](#)), TDBTransaction.EndTransaction ([329](#))

## 10.26 TDefCollection

### 10.26.1 Description

TDefCollection is a parent class for the TFieldDefs ([362](#)) and TIndexDefs ([380](#)) collections: It holds a set of named definitions on behalf of a TDataset ([285](#)) component. To this end, it introduces a dataset ([333](#)) property, and a mechanism to notify the dataset of any updates in the collection. It is supposed to hold items of class TNamedItem ([393](#)), so the TDefCollection.Find ([332](#)) method can find items by named.

### 10.26.2 Method overview

Page	Property	Description
<a href="#">332</a>	create	Instantiate a new TDefCollection instance.
<a href="#">332</a>	Find	Find an item by name
<a href="#">332</a>	GetItemNames	Return a list of all names in the collection
<a href="#">332</a>	IndexOf	Find location of item by name

### 10.26.3 Property overview

Page	Property	Access	Description
<a href="#">333</a>	Dataset	r	Dataset this collection manages definitions for.
<a href="#">333</a>	Updated	rw	Has one of the items been changed

#### 10.26.4 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 \(285\)](#), [TNamedItem \(393\)](#)

#### 10.26.5 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 \(393\)](#), [TDefCollection.IndexOf \(332\)](#)

#### 10.26.6 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 \(393\)](#)

#### 10.26.7 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 \(332\)](#), [TNamedItem.Name \(393\)](#)

### 10.26.8 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.

### 10.26.9 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.

## 10.27 TDetailedDataLink

### 10.27.1 Description

TDetailedDataLink 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 descendants.

See also: [TDataset \(285\)](#), [TDatasource \(322\)](#)

### 10.27.2 Property overview

Page	Property	Access	Description
<a href="#">333</a>	DetailDataSet	r	Detail dataset in Master-detail relation

### 10.27.3 TDetailedDataLink.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 TDetailedDataLink and is only filled in in descendent classes like TMaster-Datalink ([388](#)). The master dataset is available through the regular TDataLink.DataSource ([284](#)) property.

See also: [TDataset \(285\)](#), [TMasterDatalink \(388\)](#), [TDataLink.DataSource \(284\)](#)

## 10.28 TField

### 10.28.1 Description

`TField` is an abstract class that defines access methods for a field in a record, controlled by a `TDataset` (285) 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` (285), `TFieldDef` (358), `TFields` (365)

### 10.28.2 Method overview

Page	Property	Description
<a href="#">337</a>	Assign	Copy properties from one <code>TField</code> instance to another
<a href="#">337</a>	AssignValue	Assign value of a variant record to the field.
<a href="#">338</a>	Clear	Clear the field contents.
<a href="#">337</a>	Create	Create a new <code>TField</code> instance
<a href="#">337</a>	Destroy	Destroy the <code>TField</code> instance
<a href="#">338</a>	FocusControl	Set focus to the first control connected to this field.
<a href="#">338</a>	GetData	Get the data from this field
<a href="#">339</a>	IsBlob	Is the field a BLOB field (untyped data of indeterminate size).
<a href="#">339</a>	IsValidChar	Check whether a character is valid input for the field
<a href="#">339</a>	RefreshLookupList	Refresh the lookup list
<a href="#">339</a>	SetData	Save the field data
<a href="#">340</a>	SetFieldType	Set the field data type
<a href="#">340</a>	Validate	Validate the data buffer



### 10.28.3 Property overview

Page	Property	Access	Description
351	Alignment	rw	Alignment for this field
340	AsBCD	rw	Access the field's contents as a BCD (Binary coded Decimal)
341	AsBoolean	rw	Access the field's contents as a Boolean value.
341	AsBytes	rw	Retrieve the contents of the field as an array of bytes
341	AsCurrency	rw	Access the field's contents as a Currency value.
342	AsDateTime	rw	Access the field's contents as a TDateTime value.
342	AsFloat	rw	Access the field's contents as a floating-point (Double) value.
343	AsInteger	rw	Access the field's contents as a 32-bit signed integer (longint) value.
343	AsLargeInt	rw	Access the field's contents as a 64-bit signed integer (longint) value.
342	AsLongint	rw	Access the field's contents as a 32-bit signed integer (longint) value.
343	AsString	rw	Access the field's contents as an AnsiString value.
344	AsVariant	rw	Access the field's contents as a Variant value.
344	AsWideString	rw	Access the field's contents as a WideString value.
344	AttributeSet	rw	Not used: dictionary information
345	Calculated	rw	Is the field a calculated field ?
345	CanModify	r	Can the field's contents be modified.
352	ConstraintErrorMessage	rw	Message to display if the CustomConstraint constraint is violated.
345	CurValue	r	Current value of the field
351	CustomConstraint	rw	Custom constraint for the field's value
345	DataSet	rw	Dataset this field belongs to
346	DataSize	r	Size of the field's data
346	DataType	r	The data type of the field.
352	DefaultExpression	rw	Default value for the field
352	DisplayLabel	rws	Name of the field for display purposes
346	DisplayName	r	User-readable fieldname
346	DisplayText	r	Formatted field value
352	DisplayWidth	rw	Width of the field in characters
347	EditMask	rw	Specify an edit mask for an edit control
347	EditMaskPtr	r	Alias for EditMask
353	FieldKind	rw	The kind of field.
353	FieldName	rw	Name of the field
347	FieldNo	r	Number of the field in the record
353	HasConstraints	r	Does the field have any constraints defined
354	ImportedConstraint	rw	Constraint for the field value on the level of the underlying database
354	Index	rw	Index of the field in the list of fields
348	IsIndexField	r	Is the field an indexed field ?
348	IsNull	r	Is the field empty
354	KeyFields	rw	Key fields to use when looking up a field value.
348	Lookup	rw	Is the field a lookup field
354	LookupCache	rw	Should lookup values be cached
355	LookupDataSet	rw	Dataset with lookup values
355	LookupKeyFields	rw	Names of fields on which to perform a locate
351	LookupList	r	List of lookup values
355	LookupResultField	rw	Name of field to use as lookup value
348	NewValue	rw	The new value of the field
349	Offset	r	Offset of the field's value in the dataset buffer
350	OldValue	r	Old value of the field
357	OnChange	rw	Event triggered when the field's value has changed
358	OnGetText	rw	Event to format the field's content
358	OnSetText	rw	Event to set the field's content based on a user-formatted string
358	OnValidate	rw	Event to validate the value of a field before it is written

#### **10.28.4 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 (303) method will create the necessary TField instances, if none had been created in the designer.

**See also:** TDataset.Open (303)

#### **10.28.5 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 (293)

#### **10.28.6 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 (350)

#### **10.28.7 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` (248) exception.

See also: `TField.Assign` (337), `TField.Value` (350)

### 10.28.8 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` (348) 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` (248) exception.

See also: `TField.IsNull` (348), `TField.Value` (350)

### 10.28.9 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` (296), `TDataSet.DisableControls` (295)

### 10.28.10 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` (348), `TField.SetData` (339), `TField.Value` (350)

### 10.28.11 TField.IsBlob

**Synopsis:** Is the field a BLOB field (untyped data of indeterminate size).

**Declaration:** class function IsBlob; 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 ([262](#))

### 10.28.12 TField.IsValidChar

**Synopsis:** Check whether a character is valid input for the field

**Declaration:** function IsValidChar(InputChar: Char) : Boolean; Virtual

**Visibility:** public

**Description:** IsValidChar checks whether InputChar is a valid characters for the current field. It does this by checking whether InputChar is in the set of characters sepcified by the TField.ValidChars ([350](#)) property. The ValidChars property will be initialized to a correct set of characters by descendent classes. For instance, a numerical field will only accept numerical characters and the sign and decimal separator characters.

Descendent classes can override this method to provide custom checks. The ValidChars property can be set to restrict the list of valid characters to a subset of what would normally be available.

**See also:** TField.ValidChars ([350](#))

### 10.28.13 TField.RefreshLookupList

**Synopsis:** Refresh the lookup list

**Declaration:** procedure RefreshLookupList

**Visibility:** public

**Description:** RefreshLookupList fills the lookup list for a lookup fields with all key, value pairs found in the lookup dataset. It will open the lookup dataset if needed. The lookup list is only used if the TField.LookupCache ([354](#)) 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 ([355](#)), LookupKeyFields ([355](#)), LookupResultField ([355](#))

### 10.28.14 TField.SetData

**Synopsis:** Save the field data

**Declaration:** procedure SetData(Buffer: Pointer); Overload  
procedure SetData(Buffer: Pointer; NativeFormat: Boolean); Overload

**Visibility:** public

**Description:** SetData saves the value of the field data in Buffer to the dataset internal buffer. The Buffer pointer should point to a memory buffer containing the data for the field in the correct format. If the NativeFormat parameter is true, then date/time formats should use the TDateTime format.

There should normally not be any need to call SetData directly: it is called by the various setter methods of the AsXXX properties of TField.

**Errors:** No validity checks are performed on Buffer: it should point to a valid memory area, and should be large enough to contain the value of the field. Failure to provide a buffer that matches these criteria will result in an exception.

See also: TField.GetData ([338](#)), TField.Value ([350](#))

### 10.28.15 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 ([346](#))

### 10.28.16 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 ([358](#)) event handler, if one is set, to allow the application programmer to program additional checks.

See also: TField.SetData ([339](#)), TField.OnValidate ([358](#))

### 10.28.17 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 ([341](#)), TField.Value ([350](#))

### **10.28.18 TField.AsBoolean**

**Synopsis:** Access the field's contents as a Boolean value.

**Declaration:** Property AsBoolean : Boolean

**Visibility:** public

**Access:** Read,Write

**Description:** AsBoolean can be used to read or write the contents of the field as a boolean value. If the native type of the field is not Boolean, then an attempt will be made to convert the field value from the native format to a boolean value when reading the field's content. Likewise, when writing the property, the value will be converted to the native type of the field (if the value allows it). Therefor, when reading or writing a field value for a field whose native data type is not a Boolean value (for instance a string value), an exception may be raised.

See also: [TField.Value \(350\)](#), [TField.AsInteger \(343\)](#)

### **10.28.19 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 \(261\)](#)

### **10.28.20 TField.AsCurrency**

**Synopsis:** Access the field's contents as a Currency value.

**Declaration:** Property AsCurrency : Currency

**Visibility:** public

**Access:** Read,Write

**Description:** AsBoolean can be used to read or write the contents of the field as a currency value. If the native type of the field is not Boolean, then an attempt will be made to convert the field value from the native format to a currency value when reading the field's content. Likewise, when writing the property, the value will be converted to the native type of the field (if the value allows it). Therefor, when reading or writing a field value for a field whose native data type is not a currency-compatible value (dates or string values), an exception may be raised.

See also: [TField.Value \(350\)](#), [TField.AsFloat \(342\)](#)

### 10.28.21 TField.AsDateTime

**Synopsis:** Access the field's contents as a TDateTime value.

**Declaration:** Property AsDateTime : TDateTime

**Visibility:** public

**Access:** Read,Write

**Description:** AsDateTime can be used to read or write the contents of the field as a TDateTime value (for both date and time values). If the native type of the field is not a date or time value, then an attempt will be made to convert the field value from the native format to a TDateTime value when reading the field's content. Likewise, when writing the property, the value will be converted to the native type of the field (if the value allows it). Therefor, when reading or writing a field value for a field whose native data type is not a TDateTime-compatible value (dates or string values), an exception may be raised.

See also: TField.Value ([350](#)), TField.AsString ([343](#))

### 10.28.22 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). Therefor, when reading or writing a field value for a field whose native data type is not a floating-point-compatible value (string values for instance), an exception may be raised.

See also: TField.Value ([350](#)), TField.AsString ([343](#)), TField.AsCurrency ([341](#))

### 10.28.23 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). Therefor, when reading or writing a field value for a field whose native data type is not a 32-bit signed integer-compatible value (string values for instance), an exception may be raised.

This is an alias for the TField.AsInteger ([343](#)).

See also: TField.Value ([350](#)), TField.AsString ([343](#)), TField.AsInteger ([343](#))

### 10.28.24 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). Therefor, when reading or writing a field value for a field whose native data type is not a 64-bit signed integer-compatible value (string values for instance), an exception may be raised.

See also: TField.Value (350), TField.AsString (343), TField.AsInteger (343)

### 10.28.25 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 a integer value when reading the field's content. Likewise, when writing the property, the value will be converted to the native type of the field (if the value allows it). Therefor, when reading or writing a field value for a field whose native data type is not a 32-bit signed integer-compatible value (string values for instance), an exception may be raised.

See also: TField.Value (350), TField.AsString (343), TField.AsLongint (342), TField.AsInt64 (334)

### 10.28.26 TField.AsString

**Synopsis:** Access the field's contents as an AnsiString value.

**Declaration:** PropertyAsString : string

**Visibility:** public

**Access:** Read,Write

**Description:**AsString can be used to read or write the contents of the field as an AnsiString value. If the native type of the field is not an ansistring value, then an attempt will be made to convert the field value from the native format to a ansistring value when reading the field's content. Likewise, when writing the property, the value will be converted to the native type of the field (if the value allows it). Therefor, when reading or writing a field value for a field whose native data type is not an ansistring-compatible value, an exception may be raised.

See also: TField.Value (350), TField.AsWideString (344)

### **10.28.27 TField.AsWideString**

**Synopsis:** Access the field's contents as a WideString value.

**Declaration:** Property AsWideString : WideString

**Visibility:** public

**Access:** Read,Write

**Description:** AsWideString can be used to read or write the contents of the field as a WideString value. If the native type of the field is not a widestring value, then an attempt will be made to convert the field value from the native format to a widestring value when reading the field's content. Likewise, when writing the property, the value will be converted to the native type of the field (if the value allows it). Therefor, when reading or writing a field value for a field whose native data type is not a widestring-compatible value, an exception may be raised.

See also: TField.Value ([350](#)), TField.Astring ([334](#))

### **10.28.28 TField.AsVariant**

**Synopsis:** Access the field's contents as a Variant value.

**Declaration:** Property AsVariant : variant

**Visibility:** public

**Access:** Read,Write

**Description:** AsVariant can be used to read or write the contents of the field as a Variant value. If the native type of the field is not a Variant value, then an attempt will be made to convert the field value from the native format to a variant value when reading the field's content. Likewise, when writing the property, the value will be converted to the native type of the field (if the value allows it). Therefor, when reading or writing a field value for a field whose native data type is not a variant-compatible value, an exception may be raised.

See also: TField.Value ([350](#)), TField.Astring ([334](#))

### **10.28.29 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.

### 10.28.30 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 (353) 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 (353)

### 10.28.31 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 (356), TDataset.CanModify (307)

### 10.28.32 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 (350)

### 10.28.33 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 it 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 (285), TDataset.Fields (312)

### **10.28.34 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 (349) property which declares a logical size for datatypes that have a variable size (such as string fields). For BLOB fields, use the TBlobField.BlobSize (263) property to get the size of the field's contents for the current record..

See also: TField.Size (349), TBlobField.BlobSize (263)

### **10.28.35 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 (353)

### **10.28.36 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 (353) property, unless assigned another value.

The use of this property is deprecated. Use DisplayLabel (352) instead.

See also: Tfield.FieldName (353)

### **10.28.37 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` (343) property, which does not take into account any formatting.

See also: `TField.AsString` (343)

### 10.28.38 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` (328), `TStringField.EditMask` (415)

### 10.28.39 TField.EditMaskPtr

**Synopsis:** Alias for `EditMask`

**Declaration:** Property `EditMaskPtr` : `TEditMask`

**Visibility:** public

**Access:** Read

**Description:** `EditMaskPtr` is a read-only alias for the `EditMask` (347) property. It is not used.

See also: `TField.EditMask` (347)

### 10.28.40 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` (354)

#### 10.28.41 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 (285) can change this to True.

**See also:** TField.Calculated (345)

#### 10.28.42 TField.IsNull

**Synopsis:** Is the field empty

**Declaration:** PropertyIsNull : 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 (300) is called or Clear (338) is called then IsNull will return True.

**See also:** TField.Clear (338), TDataset.Insert (300)

#### 10.28.43 TField.Lookup

**Synopsis:** Is the field a lookup field

**Declaration:** Property Lookup : Boolean

**Visibility:** public

**Access:** Read,Write

**Description:** Lookup is True if the FieldKind (353) equals fkLookup, False otherwise. Setting the Lookup property will switch the FieldKind between the fkLookup and fkData.

**See also:** TField.FieldKind (353)

#### 10.28.44 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 (285) does not yet support this.

**See also:** TField.Value (350), TField.CurValue (345)

### **10.28.45 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 (347), TField.Index (354), TField.Datasize (346)

### **10.28.46 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 (346) should be used. For blob fields, the current size of the data is not guaranteed to be present.

**See also:** DataSize (346)

### **10.28.47 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 (358) event handler if a handler was specified. For display purposes, the TField.DisplayText (346) 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 (343) property, this will invoke the Tfield.OnSetText (358) event handler, if one is set.

**See also:** TField.AsString (343), TField.DisplayText (346), TField.Value (350)

### 10.28.48 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 ([339](#)) function.

See also: TField.IsValidChar ([339](#))

### 10.28.49 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 ([348](#)) 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 ([348](#)), TField.Text ([349](#)), TField.DisplayText ([346](#))

### 10.28.50 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 ([350](#)), TField.CurValue ([345](#)), TField.NewValue ([348](#))

### **10.28.51 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 (354) property is set to True. It can be refreshed using the RefreshLookupList (339) method.

See also: TField.RefreshLookupList (339), TField.LookupCache (354)

### **10.28.52 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 (346)

### **10.28.53 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 its 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 (352) as a message.

This feature is not yet implemented in FPC.

See also: TField.ConstraintErrorMessage (352), TField.ImportedConstraint (354)

### 10.28.54 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 ([351](#)) is violated.

This feature is not yet implemented in FPC.

See also: TField.CustomConstraint ([351](#))

### 10.28.55 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 ([291](#)) or TDataset.Insert ([300](#)) 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 ([300](#)), TDataset.Append ([291](#)), TDataset.CustomConstraint ([285](#))

### 10.28.56 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 ([353](#)) property, unless assigned another value.

See also: TField.FieldName ([353](#))

### 10.28.57 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` (349) for string fields) but can be modified to obtain a more appropriate value for the field's expected content.

See also: `TField.Alignment` (351), `TField.DisplayText` (346)

### **10.28.58 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.

### **10.28.59 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` (358), `TField.Size` (349), `TField.DataType` (346)

### **10.28.60 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` (351) or `ImportedConstraint` (354) properties is set to a non-empty value.

See also: `CustomConstraint` (351), `ImportedConstraint` (354)

### **10.28.61 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 datas structures, but this need not always be so. The TField.FieldNo ([347](#)) property should be used for that.

See also: TField.FieldNo ([347](#))

### **10.28.62 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 ([285](#)) descendent.

See also: TField.CustomConstraint ([351](#)), TDataset ([285](#)), TField.ConstraintErrorMessage ([352](#))

### **10.28.63 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 ([355](#)) in the dataset pointed to by the LookupDataset ([355](#)) property.

See also: LookupKeyFields ([355](#)), LookupDataset ([355](#))

### **10.28.64 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` (355) in the dataset pointed to by the `LookupDataset` (355) property. The list of key, value pairs is available through the `TField.LookupList` (351) property.

**See also:** `LookupKeyFields` (355), `LookupDataset` (355), `TField.LookupList` (351)

### **10.28.65 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` (355) property is used as a list of fields to locate a record in this dataset, and the value of the `LookupResultField` (355) field is then used as the value of the lookup field.

**See also:** `KeyFields` (354), `LookupKeyFields` (355), `LookupResultField` (355), `LookupCache` (354)

### **10.28.66 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` (355) property. These fields will be used when locating a record corresponding to the values in the `TField.KeyFields` (354) property.

**See also:** `KeyFields` (354), `LookupDataset` (355), `LookupResultField` (355), `LookupCache` (354)

### **10.28.67 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` (355) 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` (354), `LookupDataset` (355), `LookupKeyFields` (355), `LookupCache` (354)

### **10.28.68 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 (285) descendent or the database engine support retrieval of this property. It can be used to automatically create update statements, together with the TField.ProviderFlags (356) property.

**See also:** TDataset (285), TField.ProviderFlags (356)

### **10.28.69 TField.ProviderFlags**

**Synopsis:** Flags for provider or update support

**Declaration:** Property ProviderFlags : TProviderFlags

**Visibility:** published

**Access:** Read,Write

**Description:** ProviderFlags contains a set of flags that can be used by engines that automatically generate update SQL statements or update data packets. The various items in the set tell the engine whether the key is a key field, should be used in the where clause of an update statement or whether - in fact - it should be updated at all.

These properties should be set by the programmer so engines such as SQLDB can create correct update SQL statements whenever they need to post changes to the database. Note that to be able to set these properties in a designer, persistent fields must be created.

**See also:** TField.Origin (356)

### **10.28.70 TField.ReadOnly**

**Synopsis:** Is the field read-only

**Declaration:** Property ReadOnly : Boolean

**Visibility:** published

**Access:** Read,Write

**Description:** ReadOnly can be set to True to prevent controls of writing data to the field, effectively making it a read-only field. Setting this property to True does not prevent the field from getting a value through code: it is just an indication for GUI controls that the field's value is considered read-only.

**See also:** TFieldDef.Attributes (361)

### 10.28.71 TField.Required

**Synopsis:** Does the field require a value

**Declaration:** Property Required : Boolean

**Visibility:** published

**Access:** Read,Write

**Description:** Required determines whether the field needs a value when posting the data: when a dataset posts the changed made to a record (new or existing), it will check whether all fields with the Required property have a value assigned to them. If not, an exception will be raised. Descendents of TDataset (285) 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 (303), ReadOnly (356), Visible (357)

### 10.28.72 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 (356), TField.Required (357)

### 10.28.73 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 (358) 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 (357)

### 10.28.74 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 (349) or TField.DisplayText (346) 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 (349), TField.DisplayText (346), TField.OnSetText (358), TFieldGetTextEvent (237)

### 10.28.75 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 (349) 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 (358) handler, if it exists.

See also: TField.Text (349), TField.OnGetText (358), TFieldGetTextEvent (237)

### 10.28.76 TField.OnValidate

**Synopsis:** Event to validate the value of a field before it is written to the data buffer

**Declaration:** Property OnValidate : TFieldNotifyEvent

**Visibility:** published

**Access:** Read,Write

**Description:** OnValidate is called prior to writing a new field value to the dataset's data buffer. It can be used to prevent writing the new value to the buffer by raising an exception in the event handler. Note that this event handler is always called, irrespective of the way the value of the field is set.

See also: TField.Text (349), TField.OnGetText (358), TField.OnSetText (358), TField.OnChange (357)

## 10.29 TFieldDef

### 10.29.1 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 (309), TFieldDefs (362)

### 10.29.2 Method overview

Page	Property	Description
360	Assign	Assign the contents of one TFieldDef instance to another.
359	Create	Constructor for TFieldDef.
360	CreateField	Create TField instance based on definitions in current TFieldDef instance.
359	Destroy	Free the TFieldDef instance

### 10.29.3 Property overview

Page	Property	Access	Description
361	Attributes	rw	Additional attributes of the field.
361	DataType	rw	Data type for the field
360	FieldClass	r	TField class used for this fielddef
360	FieldNo	r	Field number
361	InternalCalcField	rw	Is this a definition of an internally calculated field ?
362	Precision	rw	Precision used in BCD (Binary Coded Decimal) fields
361	Required	rw	Is the field required ?
362	Size	rw	Size of the buffer needed to store the data of the field

### 10.29.4 TFieldDef.Create

**Synopsis:** Constructor for TFieldDef.

**Declaration:** constructor create(ACollection: TCollection); Override  
constructor Create(AOwner: TFieldDefs; const AName: string;  
ADataType: TFieldType; ASize: Integer;  
ARequired: Boolean; AFieldNo: LongInt); 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 (358), datatype (361), size (362), precision (362), FieldNo (360) and the Required (361) property are all set according to the passed arguments.

**Errors:** If a duplicate name is passed, then an exception will occur.

**See also:** Name (358), datatype (361), size (362), precision (362), FieldNo (360), Required (361)

### 10.29.5 TFieldDef.Destroy

**Synopsis:** Free the TFieldDef instance

**Declaration:** destructor Destroy; Override

**Visibility:** public

**Description:** Destroy destroys the TFieldDef instance. It simply calls the inherited destructor.

**See also:** TFieldDef.Create (359)

### 10.29.6 TFieldDef.Assign

**Synopsis:** Assign the contents of one TFieldDef instance to another.

**Declaration:** procedure Assign(APersistent: TPersistent);   Override

**Visibility:** public

**Description:** Assign assigns all published properties of APersistent to the current instance, if APersistent is an instance of class TFieldDef.

**Errors:** If APersistent is not of class TFieldDef (358), then an exception will be raised.

### 10.29.7 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 (361) what TField (334) descendent it should create, and then returns a newly created instance of this class. It sets the appropriate defaults for the Size (349), FieldName (353), FieldNo (347), Precision (334), ReadOnly (356) and Required (357) 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 (231) array is used to determine which TField Descendent class should be used when creating the TField instance, but descendants of TDataset may override the values in that array.

**See also:** DefaultFieldClasses (231), TField (334)

### 10.29.8 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 (360) 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 (285), CreateField (360), TField (334)

### 10.29.9 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.

### **10.29.10 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 rovided by the underlying data mechanism.

### **10.29.11 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.

### **10.29.12 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 ([361](#))

### **10.29.13 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 \(360\)](#), [TFieldDef.CreateField \(360\)](#)

### 10.29.14 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 \(362\)](#)

### 10.29.15 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 \(362\)](#), [TFieldDef.DataType \(361\)](#)

## 10.30 TFieldDefs

### 10.30.1 Description

`TFieldDefs` is used by each `TDataSet` instance to keep a description of the data that it manages; for each field in a record that makes up the underlying data, the `TFieldDefs` instance keeps an instance of `TFieldDef` that describes the field's contents. For any internally calculated fields of the dataset, a `TFieldDef` instance is kept as well. This collection is filled by descendent classes of `TDataSet` as soon as the dataset is opened; it is cleared when the dataset closes. After the collection was populated, the dataset creates `TField` instances based on all the definitions in the collections. If persistent fields were used, the contents of the `fielddefs` collection is compared to the field components that are present in the dataset. If the collection contains more field definitions than Field components, these extra fields will not be available in the dataset.

See also: [TFieldDef \(358\)](#), [TDataSet \(285\)](#)

### 10.30.2 Method overview

Page	Property	Description
363	Add	Add a new field definition to the collection.
363	AddFieldDef	Add new TFieldDef
364	Assign	Copy all items from one dataset to another
363	Create	Create a new instance of TFieldDefs
364	Find	Find item by name
364	MakeNameUnique	Create a unique field name starting from a base name
364	Update	Force update of definitions

### 10.30.3 Property overview

Page	Property	Access	Description
365	HiddenFields	rw	Should field instances be created for hidden fields
365	Items	rw	Indexed access to the fielddef instances

### 10.30.4 TFieldDefs.Create

Synopsis: Create a new instance of TFieldDefs

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 (358), TDataset (285)

### 10.30.5 TFieldDefs.Add

Synopsis: Add a new field definition to the collection.

Declaration: procedure Add(const AName: string; ADataType: TFieldType; ASize: Word;  
                  ARequired: Boolean); Overload  
procedure Add(const AName: string; ADataType: TFieldType; ASize: Word;  
                  ; Overload  
procedure Add(const AName: string; ADataType: TFieldType); Overload

Visibility: public

Description: Add adds a new item to the collection and fills in the Name, DataType, Size and Required properties of the newly added item with the provided parameters.

Errors: If an item with name AName already exists in the collection, then an exception will be raised.

See also: TFieldDefs.AddFieldDef (363)

### 10.30.6 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 ([363](#))

### 10.30.7 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 ([360](#))

### 10.30.8 TFieldDefs.Find

**Synopsis:** Find item by name

**Declaration:** function Find(const AName: string) : TFieldDef

**Visibility:** public

**Description:** Find simply calls the inherited TDefCollection.Find ([332](#)) to find an item with name AName and typecasts the result to TFieldDef.

**See also:** TDefCollection.Find ([332](#)), TNamedItem.Name ([393](#))

### 10.30.9 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 ([333](#))

### 10.30.10 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 ([364](#)), TFieldDef.Name ([358](#))

### 10.30.11 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 (361)

### 10.30.12 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 (358)

## 10.31 TFields

### 10.31.1 Description

TFields mimics a TCollection class for the Fields (312) property of TDataset (285) instance. Since TField (334) is a descendent of TComponent, it cannot be an item of a collection, and must be managed by another class.

See also: TField (334), TDataset (285), TDataset.Fields (312)

### 10.31.2 Method overview

Page	Property	Description
366	Add	Add a new field to the list
366	CheckFieldName	Check field name for duplicate entries
367	CheckFieldNames	Check a list of field names for duplicate entries
367	Clear	Clear the list of fields
366	Create	Create a new instance of TFields
366	Destroy	Free the TFields instance
367	FieldByName	Find a field based on its name
368	FieldByNumber	Search field based on its fieldnumber
367	FindField	Find a field based on its name
368	GetEnumerator	Return an enumerator for the for..in construct
368	GetFieldNames	Get the list of fieldnames
368	IndexOf	Return the index of a field instance
369	Remove	Remove an instance from the list

### 10.31.3 Property overview

Page	Property	Access	Description
<a href="#">369</a>	Count	r	Number of fields in the list
<a href="#">369</a>	Dataset	r	Dataset the fields belong to
<a href="#">369</a>	Fields	rw	Indexed access to the fields in the list

### 10.31.4 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 ([369](#)) property, and initializes an internal list object to store the list of fields.

See also: [TDataSet \(285\)](#), [TFields.Dataset \(369\)](#), [TField \(334\)](#)

### 10.31.5 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 ([369](#)). After that it cleans up the internal structures and then calls the inherited destructor.

See also: [TDataSet \(285\)](#), [TField \(334\)](#), [TFields.Clear \(367\)](#)

### 10.31.6 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 ([334](#)) 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 \(334\)](#), [TFields.Clear \(367\)](#)

### 10.31.7 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 ([365](#))). If it does, then an EDatabaseError ([248](#)) exception is raised.

See also: [TField.FieldName \(353\)](#), [TFields.Find \(365\)](#)

### 10.31.8 **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 ([366](#)).

**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 ([353](#)), TFields.CheckFieldName ([366](#)), TFields.Find ([365](#))

### 10.31.9 **TFields.Clear**

**Synopsis:** Clear the list of fields

**Declaration:** procedure Clear

**Visibility:** public

**Description:** Clear removes all TField ([334](#)) var instances from the list. All field instances are freed after they have been removed from the list.

**See also:** TField ([334](#))

### 10.31.10 **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 ([353](#)) property matches Value. The search is performed case-insensitively. If no field instance is found, then Nil is returned.

**See also:** TFields.FieldName ([367](#))

### 10.31.11 **TFields.FieldName**

**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 ([353](#)) property matches Value. The search is performed case-insensitively.

**Errors:** If no field instance is found, then an exception is raised. If this behaviour is undesired, use TField.FindField ([334](#)), where Nil is returned if no match is found.

**See also:** TFields.FindField ([367](#)), TFields.FieldName ([365](#)), Tfields.FieldByNumber ([368](#)), TFields.IndexOf ([368](#))

### 10.31.12 **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 ([347](#)) property matches the FieldNo parameter. If no such field is found, `Nil` is returned.

**See also:** [TFields.FieldName](#) ([367](#)), [TFields.FindField](#) ([367](#)), [TFields.IndexOf](#) ([368](#))

### 10.31.13 **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 ([370](#))

**See also:** [TFieldsEnumerator](#) ([370](#)), `#rtl.system.IEnumerable` (??)

### 10.31.14 **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](#) ([353](#))

### 10.31.15 **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.FieldName](#) ([367](#)), [TFields.FieldByNumber](#) ([368](#)), [TFields.FindField](#) ([367](#))

### **10.31.16 `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` (367)

### **10.31.17 `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 (369) property are numbered from 0 to Count-1.

**See also:** `TFIELDS.fields` (369)

### **10.31.18 `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 (365) 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` (366), `TField.Dataset` (345), `TDataSet` (285)

### **10.31.19 `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.FieldName` (367)

## 10.32 TFieldsEnumerator

### 10.32.1 Description

`TFieldsEnumerator` implements all the methods of `IEnumerator` so a `TFields` (365) 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` (334), `TFields` (365), `#rtl.system.IEnumerator` (??)

### 10.32.2 Method overview

Page	Property	Description
370	Create	Create a new instance of <code>TFieldsEnumerator</code> .
370	MoveNext	Move the current field to the next field in the collection.

### 10.32.3 Property overview

Page	Property	Access	Description
371	Current	r	Return the current field

### 10.32.4 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` (365) instance that created the enumerator. It initializes the enumerator position.

### 10.32.5 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 \(371\)](#)

### 10.32.6 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 \(370\)](#)

## 10.33 TFloatField

### 10.33.1 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 ([372](#)), MaxValue ([373](#)), MinValue ([373](#)) and overrides some TField ([334](#)) methods to work with floating point data.

It should never be necessary to create an instance of TFfloatField manually, a field of this class will be instantiated automatically for each floating-point field when a dataset is opened.

See also: [Currency \(372\)](#), [MaxValue \(373\)](#), [MinValue \(373\)](#)

### 10.33.2 Method overview

Page	Property	Description
<a href="#">372</a>	CheckRange	Check whether a value is in the allowed range of values for the field
<a href="#">371</a>	Create	Create a new instance of the TFfloatField

### 10.33.3 Property overview

Page	Property	Access	Description
<a href="#">372</a>	Currency	rw	Is the field a currency field.
<a href="#">373</a>	MaxValue	rw	Maximum value for the field
<a href="#">373</a>	MinValue	rw	Minimum value for the field
<a href="#">373</a>	Precision	rw	Precision (number of digits) of the field in text representations
<a href="#">372</a>	Value	rw	Value of the field as a double type

### 10.33.4 TFfloatField.Create

**Synopsis:** Create a new instance of the TFfloatField

**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.

### 10.33.5 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 ([373](#)) and MaxValue ([373](#)) properties. If the value lies outside of the allowed range, then False is returned.

See also: [MaxValue \(373\)](#), [MinValue \(373\)](#)

### 10.33.6 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 ([342](#))

See also: [TField.AsFloat \(342\)](#), [TField.Value \(350\)](#)

### 10.33.7 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 ([346](#)) and TField.Text ([349](#)) 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 \(346\)](#), [TField.Text \(349\)](#), [TNumericField.DisplayFormat \(395\)](#), [TNumericField.EditFormat \(395\)](#)

### **10.33.8 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` (373) should also be set, because it will also be checked.

**See also:** `TFloatField.MinValue` (373)

### **10.33.9 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` (373) should also be set, because it will also be checked.

**See also:** `TFloatField.MaxValue` (373), `TFloatField.CheckRange` (372)

### **10.33.10 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` (346) or `TField.Text` (349), it is used in the arguments to `FormatFloat` (??).

**See also:** `TField.Displaytext` (346), `TField.Text` (349), `FormatFloat` (??)

## 10.34 TFMTBCDField

### 10.34.1 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 \(371\)](#)

### 10.34.2 Method overview

Page	Property	Description
<a href="#">374</a>	CheckRange	Check value if it is in the range defined by MinValue and MaxValue
<a href="#">374</a>	Create	Create a new instance of the TFMTBCDField class.

### 10.34.3 Property overview

Page	Property	Access	Description
<a href="#">375</a>	Currency	rw	Does the field contain currency data ?
<a href="#">375</a>	MaxValue	rw	Maximum value for the field
<a href="#">376</a>	MinValue	rw	Minimum value for the field
<a href="#">375</a>	Precision	rw	Total number of digits in the BCD data
<a href="#">376</a>	Size		Number of digits after the decimal point
<a href="#">375</a>	Value	rw	The value of the field as a BCD value

### 10.34.4 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 ([231](#)), MaxValue ([231](#)), Size ([349](#)) (15) and Precision ([231](#)) (2) fields to their default values.

See also: [MinValue \(231\)](#), [MaxValue \(231\)](#), [Size \(349\)](#), [Precision \(231\)](#)

### 10.34.5 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 ([231](#)) and MaxValue ([231](#)) 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 \(231\)](#), [MaxValue \(231\)](#)

### 10.34.6 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 (342), TField.AsCurrency (341)

### 10.34.7 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 (342), TField.AsCurrency (341), Size (231)

### 10.34.8 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 (334) or EditFormat (334) are set, these values are used instead to format the value.

**See also:** TField.DisplayFormat (334), TField.EditFormat (334)

### 10.34.9 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 (231) or it will not have any effect.

**See also:** TFMTBCDField.CheckRange (374), MinValue (231)

### 10.34.10 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` (231) or it will not have any effect.

**See also:** `TFMTBCDField.CheckRange` (374), `MaxValue` (231)

### 10.34.11 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` (231) property it determines the maximum allowed range of values for the field. This range can be restricted using the `MinValue` (231) and `MaxValue` (231) properties.

**See also:** `MinValue` (231), `MaxValue` (231), `Precision` (231)

## 10.35 TGraphicField

### 10.35.1 Description

`TGraphicField` is the class used when a dataset must manage graphical BLOB data. (`TField.DataType` (346) equals `ftGraphic`). It initializes some of the properties of the `TField` (334) class. All methods to be able to work with graphical BLOB data have been implemented in the `TBlobField` (261) 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` (285), `TField` (334), `TLOBField` (261), `TMemoField` (392), `TWideMemoField` (417)

### 10.35.2 Method overview

Page	Property	Description
376	Create	Create a new instance of the <code>TGraphicField</code> class

### 10.35.3 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` (334) properties to configure the instance for working with graphical BLOB values.

See also: `TField` (334)

## 10.36 `TGuidField`

### 10.36.1 Description

`TGUIDField` is the class used when a dataset must manage native variant-typed data. (`TField.DataType` (346) equals `ftGUID`). It initializes some of the properties of the `TField` (334) 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` (285), `TField` (334), `TGuidField.AsGuid` (377)

### 10.36.2 Method overview

Page	Property	Description
377	Create	Create a new instance of the <code>TGUIDField</code> class

### 10.36.3 Property overview

Page	Property	Access	Description
377	AsGuid	rw	Field content as a GUID value

### 10.36.4 `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` (334) properties to configure the instance for working with GUID values.

See also: `TField` (334)

### 10.36.5 `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` (343)

## 10.37 TIndexDef

### 10.37.1 Description

TIndexDef describes one index in a set of indexes of a TDataset (285) instance. The collection of indexes is described by the TIndexDefs (380) class. It just has the necessary properties to describe an index, but does not implement any functionality to maintain an index.

See also: TIndexDefs (380)

### 10.37.2 Method overview

Page	Property	Description
378	Create	Create a new index definition

### 10.37.3 Property overview

Page	Property	Access	Description
379	CaseInsFields	rw	Fields in field list that are ordered case-insensitively
379	DescFields	rw	Fields in field list that are ordered descending
378	Expression	rw	Expression that makes up the index values
379	Fields	rw	Fields making up the index
380	Options	rw	Index options
380	Source	rw	Source of the index

### 10.37.4 TIndexDef.Create

Synopsis: Create a new index definition

Declaration: constructor Create(Owner: TIndexDefs; const AName: string;  
const TheFields: string; TheOptions: TIndexOptions)  
; Overload

Visibility: public

Description: Create initializes a new TIndexDef (378) 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 (380) 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 (380), TIndexDef.Options (380), TIndexDef.Fields (379)

### 10.37.5 TIndexDef.Expression

Synopsis: Expression that makes up the index values

Declaration: Property Expression : string

Visibility: public

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 (380)

See also: TIndexDef.Options (380), TindexDef.Fields (379)

### 10.37.6 TIndexDef.Fields

**Synopsis:** Fields making up the index

**Declaration:** Property Fields : string

**Visibility:** public

**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 (379) property can be used to determine the fields in the list that have a descending sort order. The TIndexDef.CaseInsFields (379) property determines which fields are sorted in a case-insensitive manner.

See also: TIndexDef.DescFields (379), TIndexDef.CaseInsFields (379), TIndexDef.Expression (378)

### 10.37.7 TIndexDef.CaseInsFields

**Synopsis:** Fields in field list that are ordered case-insensitively

**Declaration:** Property CaseInsFields : string

**Visibility:** public

**Access:** Read,Write

**Description:** CaseInsFields is a list of fieldnames, separated by semicolons. It contains the names of the fields in the Fields (379) property which are ordered in a case-insensitive manner. CaseInsFields may not contain fieldnames that do not appear in Fields.

See also: TIndexDef.Fields (379), TIndexDef.Expression (378), TIndexDef.DescFields (379)

### 10.37.8 TIndexDef.DescFields

**Synopsis:** Fields in field list that are ordered descending

**Declaration:** Property DescFields : string

**Visibility:** public

**Access:** Read,Write

**Description:** DescFields is a list of fieldnames, separated by semicolons. It contains the names of the fields in the Fields (379) property which are ordered in a descending manner. DescFields may not contain fieldnames that do not appear in Fields.

See also: TIndexDef.Fields (379), TIndexDef.Expression (378), TIndexDef.DescFields (379)

### 10.37.9 TIndexDef.Options

**Synopsis:** Index options

**Declaration:** Property Options : TIndexOptions

**Visibility:** public

**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 (241) for more information on the various available options.

See also: TIndexOptions (241)

### 10.37.10 TIndexDef.Source

**Synopsis:** Source of the index

**Declaration:** Property Source : string

**Visibility:** public

**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.

## 10.38 TIndexDefs

### 10.38.1 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 (378), TIndexDefs.Items (382)

### 10.38.2 Method overview

Page	Property	Description
381	Add	Add a new index definition with given name and options
381	AddIndexDef	Add a new, empty, index definition
381	Create	Create a new TIndexDefs instance
381	Find	Find an index by name
382	FindIndexForFields	Find index definition based on field names
382	GetIndexForFields	Get index definition based on field names
382	Update	Called whenever one of the items changes

### 10.38.3 Property overview

Page	Property	Access	Description
382	Items	rw	Indexed access to the index definitions

#### 10.38.4 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 (378).

**See also:** TIndexDef (378), TIndexDefs.Destroy (380)

#### 10.38.5 TIndexDefs.Add

**Synopsis:** Add a new index definition with given name and options

**Declaration:** procedure Add(const Name: string; const Fields: string;  
Options: TIndexOptions)

**Visibility:** public

**Description:** Add adds a new TIndexDef (378) 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 (378), TNamedItem.Name (393), TIndexDef.Fields (379), TIndexDef.Options (380),  
TIndexDefs.AddIndexDef (381)

#### 10.38.6 TIndexDefs.AddIndexDef

**Synopsis:** Add a new, empty, index definition

**Declaration:** function AddIndexDef : TIndexDef

**Visibility:** public

**Description:** AddIndexDef adds a new TIndexDef (378) 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 (381)

#### 10.38.7 TIndexDefs.Find

**Synopsis:** Find an index by name

**Declaration:** function Find(const IndexName: string) : TIndexDef

**Visibility:** public

**Description:** Find overloads the TDefCollection.Find (332) method to search and return a TIndexDef (378) 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 (378), TDefCollection.Find (332), TIndexDefs.FindIndexForFields (382)

### 10.38.8 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 ([379](#)) property matches the list of fields in Fields. If it finds an index definition, then it returns the found instance.

**Errors:** If no matching definition is found, an exception is raised. This is different from other Find functionality, where Find usually returns Nil if nothing is found.

**See also:** TIndexDef ([378](#)), TIndexDefs.Find ([381](#)), TIndexDefs.GetIndexForFields ([382](#))

### 10.38.9 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 ([379](#)) 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 ([378](#)), TIndexDefs.Find ([381](#)), TIndexDefs.FindIndexForFields ([382](#))

### 10.38.10 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.

### 10.38.11 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 ([378](#))

## 10.39 TIntegerField

### 10.39.1 Description

TIntegerField is an alias for TLongintField (385).

See also: TLongintField (385), TField (334)

## 10.40 TLargeintField

### 10.40.1 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 (334) 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 (384) and MaxValue (384).

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 (334), MinValue (384), MaxValue (384)

### 10.40.2 Method overview

Page	Property	Description
383	CheckRange	Check whether a values falls within the allowed range
383	Create	Create a new instance of the TLargeintField class

### 10.40.3 Property overview

Page	Property	Access	Description
384	MaxValue	rw	Maximum value for the field
384	MinValue	rw	Minimum value for the field
384	Value	rw	Field contents as a 64-bit integer value

### 10.40.4 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 (334) and MinValue (384) and MaxValue (384).

See also: TField (334), MinValue (384), MaxValue (384)

### 10.40.5 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 ([384](#)) and MaxValue ([384](#)) properties. If the value lies outside of the allowed range, then False is returned.

See also: MaxValue ([384](#)), MinValue ([384](#))

### 10.40.6 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 ([343](#)).

See also: TField.Value ([350](#)), TField.AsLargeInt ([343](#))

### 10.40.7 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 ([384](#)) should also be set, because it will also be checked.

See also: TLargeIntField.MinValue ([384](#))

### 10.40.8 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 ([384](#)) should also be set, because it will also be checked.

See also: TLargeIntField.MaxValue ([384](#))

## 10.41 TLongintField

### 10.41.1 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` (334) 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` (386) and `MaxValue` (386).

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` (334), `MaxValue` (386), `MinValue` (386)

### 10.41.2 Method overview

Page	Property	Description
385	<code>CheckRange</code>	Check whether a valid is in the allowed range of values for the field
385	<code>Create</code>	Create a new instance of <code>TLongintField</code>

### 10.41.3 Property overview

Page	Property	Access	Description
386	<code>MaxValue</code>	<code>rw</code>	Maximum value for the field
386	<code>MinValue</code>	<code>rw</code>	Minimum value for the field
386	<code>Value</code>	<code>rw</code>	Value of the field as longint

### 10.41.4 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` (386) and `MaxValue` (386) properties.

See also: `TField` (334), `MaxValue` (386), `MinValue` (386)

### 10.41.5 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` (386) and `MaxValue` (386) properties. If the value lies outside of the allowed range, then `False` is returned.

See also: `MaxValue` (386), `MinValue` (386)

### **10.41.6 `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` ([343](#)) property.

**See also:** `TField.Value` ([350](#))

### **10.41.7 `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` ([386](#))

### **10.41.8 `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` ([386](#))

## **10.42 `TLookupList`**

### **10.42.1 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` ([354](#))

### 10.42.2 Method overview

Page	Property	Description
387	Add	Add a key, value pair to the list
387	Clear	Remove all key, value pairs from the list
387	Create	Create a new instance of TLookupList.
387	Destroy	Free a TLookupList instance from memory
388	FirstKeyByValue	Find the first key that matches a value
388	ValueOfKey	Look up value based on a key
388	ValuesToStrings	Convert values to stringlist

### 10.42.3 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 (387)

### 10.42.4 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 (387)

### 10.42.5 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 (387)

### 10.42.6 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 (387)

### 10.42.7 TLookupList.FirstKeyByValue

**Synopsis:** Find the first key that matches a value

**Declaration:** function FirstKeyByValue(const AValue: Variant) : Variant

**Visibility:** public

**Description:** FirstKeyByValue does a reverse lookup: it returns the first key value in the list that matches the AValue value. If none is found, Null is returned. This mechanism is quite slow, as a linear search is performed.

**Errors:** If no key is found, Null is returned.

**See also:** TLookupList.ValueOfKey (388)

### 10.42.8 TLookupList.ValueOfKey

**Synopsis:** Look up value based on a key

**Declaration:** function ValueOfKey(const AKey: Variant) : Variant

**Visibility:** public

**Description:** ValueOfKey does a value lookup based on a key: it returns the value in the list that matches the AKey key. If none is found, Null is returned. This mechanism is quite slow, as a linear search is performed.

**See also:** TLookupList.FirstKeyByValue (388), TLookupList.Add (387)

### 10.42.9 TLookupList.ValuesToStrings

**Synopsis:** Convert values to stringlist

**Declaration:** procedure ValuesToStrings(AStrings: TStrings)

**Visibility:** public

**Description:** ValuesToStrings converts the list of values to a stringlist, so they can be used e.g. in a dropdown list.

**See also:** TLookupList.ValueOfKey (388)

## 10.43 TMasterDataLink

### 10.43.1 Description

TMasterDataLink is a TDatalink descendent which handles master-detail relations. It can be used in TDataset (285) descendants 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 (284) property.

The class also provides functionality for keeping a list of fields that make up the master-detail functionality, in the TMasterDatalink.FieldNames (389) and TMasterDataLink.Fields (390) properties.

This class should never be used in application code.

**See also:** TDataset (285), TDatalink.DataSource (284), TDatalink.DataSet (284), TMasterDatalink.FieldNames (389), TMasterDataLink.Fields (390)

### 10.43.2 Method overview

Page	Property	Description
389	Create	Create a new instance of TMasterDataLink
389	Destroy	Free the datalink instance from memory

### 10.43.3 Property overview

Page	Property	Access	Description
389	FieldNames	rw	List of fieldnames that make up the master-detail relationship
390	Fields	r	List of fields as specified in FieldNames
390	OnMasterChange	rw	Called whenever the master dataset data changes
390	OnMasterDisable	rw	Called whenever the master dataset is disabled

### 10.43.4 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 (333) property. The master dataset must be set through the DataSource (284) property, and is usually set by the application programmer.

See also: TDatalink.DetailDataset (333), TDatalink.DataSource (284)

### 10.43.5 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 (389)

### 10.43.6 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 (284)) on which the master-detail relationship is based. Setting this property will fill the TMasterDataLink.Fields (390) property with the field instances of the master dataset.

See also: TMasterDataLink.Fields (390), TDatalink.Dataset (284), TDataset.GetFieldList (299)

### 10.43.7 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 (334) instances from the master dataset (TDatalink.Dataset (284)) when the FieldNames (389) property is set, and when the master dataset opens.

**See also:** TField (334), TMasterDatalink.FieldNames (389)

### 10.43.8 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 (390) list is empty, TMasterDataLink.OnMasterDisable (390) is called instead.

**See also:** TMasterDataLink.OnMasterDisable (390)

### 10.43.9 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 (390)

## 10.44 TMasterParamsDataLink

### 10.44.1 Description

TMasterParamsDataLink is a TDatalink (280) descendent that can be used to establish a master-detail relationship between 2 TDataset instances where the detail dataset is parametrized 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 (280), TDataset (285), TParams (407), TParam (395)

### 10.44.2 Method overview

Page	Property	Description
391	CopyParamsFromMaster	Copy parameter values from master dataset.
391	Create	Initialize a new TMasterParamsDataLink instance
391	RefreshParamNames	Refresh the list of parameter names

### 10.44.3 Property overview

Page	Property	Access	Description
392	Params	rw	Parameters of detail dataset.

### 10.44.4 TMasterParamsDataLink.Create

**Synopsis:** Initialize a new TMasterParamsDataLink instance

**Declaration:** constructor Create (ADataSet: TDataSet); Override

**Visibility:** public

**Description:** Create first calls the inherited constructor using ADataSet, and then looks for a property named Params of type TParams (407) in the published properties of ADataSet and assigns it to the Params (392) property.

**See also:** TDataset (285), TParams (407), TMasterParamsDataLink.Params (392)

### 10.44.5 TMasterParamsDataLink.RefreshParamNames

**Synopsis:** Refresh the list of parameter names

**Declaration:** procedure RefreshParamNames; Virtual

**Visibility:** public

**Description:** RefreshParamNames scans the Params (392) property and sets the FieldNames (389) property to the list of parameter names.

**See also:** TMasterParamsDataLink.Params (392), TMasterDataLink.FieldNames (389)

### 10.44.6 TMasterParamsDataLink.CopyParamsFromMaster

**Synopsis:** Copy parameter values from master dataset.

**Declaration:** procedure CopyParamsFromMaster (CopyBound: Boolean); Virtual

**Visibility:** public

**Description:** CopyParamsFromMaster calls TParams.CopyParamValuesFromDataset (411), 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 (403) 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 (411), TParam.Bound (403)

### 10.44.7 TMasterParamsDataLink.Params

**Synopsis:** Parameters of detail dataset.

**Declaration:** Property Params : TParams

**Visibility:** public

**Access:** Read,Write

**Description:** Params is the TParams instance of the detail dataset. If the detail dataset contains a property named Params of type TParams, then it will be set when the TMasterParamsDataLink instance was created. If the property is not published, or has another name, then the Params property must be set in code.

See also: [Tparams \(407\)](#), [TMasterParamsDataLink.Create \(391\)](#)

## 10.45 TMemoField

### 10.45.1 Description

TMemoField is the class used when a dataset must manage memo (Text BLOB) data. (TField.DataType ([346](#)) equals ftMemo). It initializes some of the properties of the TField ([334](#)) class. All methods to be able to work with memo fields have been implemented in the TBlobField ([261](#)) 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 \(285\)](#), [TField \(334\)](#), [TBLOBField \(261\)](#), [TWideMemoField \(417\)](#), [TGraphicField \(376\)](#)

### 10.45.2 Method overview

Page	Property	Description
<a href="#">392</a>	Create	Create a new instance of the TMemoField class

### 10.45.3 Property overview

Page	Property	Access	Description
<a href="#">393</a>	Transliterate		Should the contents of the field be transliterated

### 10.45.4 TMemoField.Create

**Synopsis:** Create a new instance of the TMemoField class

**Declaration:** constructor Create(AOwner: TComponent); Override

**Visibility:** public

**Description:** Create initializes a new instance of the TMemoField class. It calls the inherited destructor, and then sets some TField ([334](#)) properties to configure the instance for working with memo values.

See also: [TField \(334\)](#)

### 10.45.5 TMemoField.Transliterate

**Synopsis:** Should the contents of the field be transliterated

**Declaration:** Property Transliterate :

**Visibility:** published

**Access:**

**Description:** Transliterate is redefined from TBlobField.Transliterate (264) with a default value of true.

**See also:** TBlobField.Transliterate (264), TStringField.Transliterate (414), TDataSet.Translate (305)

## 10.46 TNamedItem

### 10.46.1 Description

`TNamedItem` is a `TCollectionItem` (??) descendent which introduces a `Name` (393) property. It automatically returns the value of the `Name` property as the value of the `DisplayName` (393) property.

**See also:** `DisplayName` (393), `Name` (393)

### 10.46.2 Property overview

Page	Property	Access	Description
<a href="#">393</a>	<code>DisplayName</code>	<code>rw</code>	Display name
<a href="#">393</a>	<code>Name</code>	<code>rw</code>	Name of the item

### 10.46.3 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` (393) property.

**See also:** `Name` (393)

### 10.46.4 TNamedItem.Name

**Synopsis:** Name of the item

**Declaration:** Property Name : string

**Visibility:** published

**Access:** Read,Write

**Description:** `Name` is the name of the item in the collection. This property is also used as the value for the `DisplayName` (393) property. If the `TNamedItem` item is owned by a `TDefCollection` (331) collection, then the name must be unique, i.e. each `Name` value may appear only once in the collection.

**See also:** `DisplayName` (393), `TDefCollection` (331)

## 10.47 TNumericField

### 10.47.1 Description

`TNumericField` is an abstract class which overrides some of the methods of `TField` (334) 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` (395) and `TNumericField.EditFormat` (395).

Since `TNumericField` is an abstract class, it must never be instantiated directly. Instead one of the descendent classes should be created.

See also: `TField` (334), `TNumericField.DisplayFormat` (395), `TNumericField.EditFormat` (395), `TField.Alignment` (351), `TIntegerField` (383), `TLargeIntField` (383), `TFloatField` (371), `TBCDField` (257)

### 10.47.2 Method overview

Page	Property	Description
394	Create	Create a new instance of <code>TNumericField</code>

### 10.47.3 Property overview

Page	Property	Access	Description
394	Alignment		Alignment of the field
395	DisplayFormat	rw	Format string for display of numerical data
395	EditFormat	rw	Format string for editing of numerical data

### 10.47.4 TNumericField.Create

Synopsis: Create a new instance of `TNumericField`

Declaration: constructor Create(AOwner: TComponent); Override

Visibility: public

Description: `Create` calls the inherited constructor and then initializes the `TField.Alignment` (351) property with

See also: `TField.Alignment` (351)

### 10.47.5 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` (351)

### 10.47.6 **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 (346) 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 (346), TNumericField.EditFormat (395)

### 10.47.7 **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 (349) 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 (349), TNumericField.DisplayFormat (395)

## 10.48 **TParam**

### 10.48.1 **Description**

TParam is one item in a TParams (407) collection. It describes the name (TParam.Name (405)), type (ParamType (406)) and value (TParam.Value (405)) of a parameter in a parametrized query or stored procedure. Under normal circumstances, it should never be necessary to create a TParam instance manually; the TDataset (285) descendent that owns the parameters should have created all necessary TParam instances.

**See also:** TParams (407)

### 10.48.2 Method overview

Page	Property	Description
397	Assign	Assign one parameter instance to another
397	AssignField	Copy value from field instance
398	AssignFieldValue	Assign field value to the parameter.
398	AssignFromField	Copy field type and value
397	AssignToField	Assign parameter value to field
398	Clear	Clear the parameter value
396	Create	Create a new parameter value
398	GetData	Get the parameter value from a memory buffer
399	GetDataSize	Return the size of the data.
399	LoadFromFile	Load a parameter value from file
399	LoadFromStream	Load a parameter value from stream
399	SetBlobData	Set BLOB data
400	SetData	Set the parameter value from a buffer

### 10.48.3 Property overview

Page	Property	Access	Description
400	AsBlob	rw	Return parameter value as a blob
400	AsBoolean	rw	Get/Set parameter value as a boolean value
400	AsCurrency	rw	Get/Set parameter value as a currency value
401	AsDate	rw	Get/Set parameter value as a date (TDateTime) value
401	AsDateTime	rw	Get/Set parameter value as a date/time (TDateTime) value
401	AsFloat	rw	Get/Set parameter value as a floating-point value
403	AsFMTBCD	rw	Parameter value as a BCD value
401	AsInteger	rw	Get/Set parameter value as an integer (32-bit) value
402	AsLargeInt	rw	Get/Set parameter value as a 64-bit integer value
402	AsMemo	rw	Get/Set parameter value as a memo (string) value
402	AsSmallInt	rw	Get/Set parameter value as a smallint value
402	AsString	rw	Get/Set parameter value as a string value
403	AsTime	rw	Get/Set parameter value as a time (TDateTime) value
405	AsWideString	rw	Get/Set the value as a widestring
403	AsWord	rw	Get/Set parameter value as a word value
403	Bound	rw	Is the parameter value bound (set to fixed value)
404	Dataset	r	Dataset to which this parameter belongs
405	DataType	rw	Data type of the parameter
404	IsNull	r	Is the parameter empty
405	Name	rw	Name of the parameter
404	NativeStr	rw	No description available
406	NumericScale	rw	Numeric scale
406	ParamType	rw	Type of parameter
406	Precision	rw	Precision of the BCD value
407	Size	rw	Size of the parameter
404	Text	rw	Read or write the value of the parameter as a string
405	Value	rws	Value as a variant

### 10.48.4 TParam.Create

**Synopsis:** Create a new parameter value

**Declaration:** constructor Create(ACollection: TCollection); Overload  
constructor Create(AParams: TParams; AParamType: TParamType); Overload

---

; Reintroduce

**Visibility:** public

**Description:** Create first calls the inherited create, and then initializes the parameter properties. The first form creates a default parameter, the second form is a convenience function and initializes a parameter of a certain kind (AParamType), in which case the owning TParams collection must be specified in AParams

**See also:** [TParams \(407\)](#)

### 10.48.5 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 (334), then it is passed to TParam.AssignField (397). If Source is of type TStrings, then it is assigned to TParams.AsMemo (407).

**Errors:** If Source is not of type TParam, TField or TStrings, an exception will be raised.

**See also:** [TField \(334\)](#), [TParam.Name \(405\)](#), [TParam.Bound \(403\)](#), [TParam.NumericScale \(406\)](#), [TParam.ParamType \(406\)](#), [TParam.value \(405\)](#), [TParam.Size \(407\)](#), [TParam.AssignField \(397\)](#), [Tparam.AsMemo \(402\)](#)

### 10.48.6 TParam.AssignField

**Synopsis:** Copy value from field instance

**Declaration:** procedure AssignField(Field: TField)

**Visibility:** public

**Description:** AssignField copies the Field, FieldName (353) and Value (350) 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 \(397\)](#), [TParam.AssignToField \(397\)](#), [TParam.AssignFieldValue \(398\)](#)

### 10.48.7 TParam.AssignToField

**Synopsis:** Assign parameter value to field

**Declaration:** procedure AssignToField(Field: TField)

**Visibility:** public

**Description:** AssignToField copies the parameter value (405) to the field instance. If Field is Nil, nothing happens.

**Errors:** An EDatabaseError (248) exception is raised if the field has an unsupported field type (for types ftCursor, ftArray, ftDataset, ftReference).

**See also:** [TParam.Assign \(397\)](#), [TParam.AssignField \(397\)](#), [TParam.AssignFromField \(398\)](#)

### 10.48.8 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 (403) bound parameter to True. This method is called from TParam.AssignField (397).

**See also:** TField (334), TParam.AssignField (397), TParam.Bound (403)

### 10.48.9 TParam.AssignFromField

**Synopsis:** Copy field type and value

**Declaration:** procedure AssignFromField(Field: TField)

**Visibility:** public

**Description:** AssignFromField copies the field value (350) and data type (TField.DataType (346)) to the parameter instance. If Field is Nil, nothing happens. This is the reverse operation of TParam.AssignToField (397).

**Errors:** An EDatabaseError (248) exception is raised if the field has an unsupported field type (for types ftCursor, ftArray, ftDataset, ftReference).

**See also:** TParam.Assign (397), TParam.AssignField (397), TParam.AssignToField (397)

### 10.48.10 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 (405), TParam.Name (405), TParam.ParamType (406), TParam.DataType (405)

### 10.48.11 TParam.GetData

**Synopsis:** Get the parameter value from a memory buffer

**Declaration:** procedure GetData(Buffer: Pointer)

**Visibility:** public

**Description:** GetData retrieves the parameter value and stores it in buffer. It uses the same data layout as TField (334), 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 (248) exception.

**See also:** TParam.SetData (400), TField (334)

### 10.48.12 TParam.GetSize

**Synopsis:** Return the size of the data.

**Declaration:** function GetSize : Integer

**Visibility:** public

**Description:** GetSize returns the size (in bytes) needed to store the current value of the parameter.

**Errors:** For an unsupported data type, an EDatabaseError (248) exception is raised when this function is called.

**See also:** TParam.GetData (398), TParam.SetData (400)

### 10.48.13 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 (399).

**Errors:** If the specified FileName is not a valid file, or the file is not readable, an exception will occur.

**See also:** TParam.LoadFromStream (399), TBlobType (233), TParam.SaveToFile (395)

### 10.48.14 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 (399), TParam.SaveToStream (395)

### 10.48.15 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 (399)

### 10.48.16 TParam.SetData

**Synopsis:** Set the parameter value from a buffer

**Declaration:** procedure SetData(Buffer: Pointer)

**Visibility:** public

**Description:** SetData performs the rever operation of TParam.GetData (398): it reads the parameter value from the memory area pointed to by Buffer. The size of the data read is determined by TParam.GetDataSetSize (399) and the type of data by TParam.DataType (405) : it is the same storage mechanism used by TField (334), and so can be used to copy the value from a TDataset (285) record buffer.

**Errors:** Not all field types are supported. If an unsupported field type is encountered, an EDatabaseError (248) exception is raised.

**See also:** TDataset (285), TParam.GetData (398), TParam.DataType (405), TParam.GetDataSetSize (399)

### 10.48.17 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:** TParamAsString (402)

### 10.48.18 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 (405), TParam.Value (405)

### 10.48.19 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 (401), TParam.DataType (405), TParam.Value (405)

### 10.48.20 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 (401), TParam.AsTime (403), TParam.DataType (405), TParam.Value (405)

### 10.48.21 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 (401), TParam.asTime (403), TParam.DataType (405), TParam.Value (405)

### 10.48.22 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 (400), TParam.DataType (405), TParam.Value (405)

### 10.48.23 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 (402), TParam.AsSmallInt (402), TParam.AsWord (403), TParam.DataType (405), TParam.Value (405)

### 10.48.24 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 (401), TParam.asSmallint (402), TParam.AsWord (403), TParam.DataType (405), TParam.Value (405)

### 10.48.25 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 (402), TParam.LoadFromStream (399), TParam.SaveToStream (395), TParam.DataType (405), TParam.Value (405)

### 10.48.26 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 (401), TParam.AsLargeInt (402), TParam.AsWord (403), TParam.DataType (405), TParam.Value (405)

### 10.48.27 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 (405), TParam.Value (405)

### **10.48.28 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 ([401](#)), TParam.AsDateTime ([401](#)), TParam.DataType ([405](#)), TParam.Value ([405](#))

### **10.48.29 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 ([401](#)), TParam.AsLargeInt ([402](#)), TParam.AsSmallint ([402](#)), TParam.DataType ([405](#)), TParam.Value ([405](#))

### **10.48.30 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 ([231](#)), AsCurrency ([231](#))

### **10.48.31 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 ([405](#)), TParam.Value ([405](#))

### 10.48.32 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 (407) 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 (285), TParams (407)

### 10.48.33 TParam.IsNull

**Synopsis:** Is the parameter empty

**Declaration:** Property IsNull : Boolean

**Visibility:** public

**Access:** Read

**Description:** IsNull is True if the value is empty or not set (Null or UnAssigned).

**See also:** TParam.Clear (398), TParam.Value (405)

### 10.48.34 TParam.NativeStr

**Synopsis:** No description available

**Declaration:** Property NativeStr : string

**Visibility:** public

**Access:** Read,Write

**Description:** No description available

### 10.48.35 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 (402), but, when written, does not set the data type; instead, it attempts to convert the value to the type specified in TParam.DataType (405).

**See also:** TParam.AsString (402), TParam.DataType (405)

### **10.48.36 TParam.Value**

**Synopsis:** Value as a variant

**Declaration:** Property Value : Variant

**Visibility:** public

**Access:** Read,Write

**Description:** Value returns (or sets) the value as a variant value.

**See also:** TParam.DataType ([405](#))

### **10.48.37 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 ([405](#)) to ftWideString.

**See also:** TParam.AsString ([402](#)), TParam.Value ([405](#)), TParam.DataType ([405](#))

### **10.48.38 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 ([404](#)), TParam.Value ([405](#)), TParam.AssignField ([397](#))

### **10.48.39 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 ([405](#)), TParam.Value ([405](#)), TParams.ParamByName ([410](#))

#### 10.48.40 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 ([406](#)), TParam.Size ([407](#))

#### 10.48.41 TParam.ParamType

**Synopsis:** Type of parameter

**Declaration:** Property ParamType : TParamType

**Visibility:** published

**Access:** Read,Write

**Description:** ParamTyp specifies the type of parameter: is the parameter value written to the database engine, or is it received from the database engine, or both ? It can have the following value:

**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 ([405](#)), TParam.DataSize ([395](#)), TParam.Name ([405](#))

#### 10.48.42 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 ([406](#)), TParam.Size ([407](#))

### 10.48.43 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 (346) in the TParam.AssignFieldValue (398) method. The actual size can be retrieved through the TParam.Datasize (395) property.

See also: TParam.Datasize (395), TField.DataSize (346), TParam.AssignFieldValue (398)

## 10.49 TParams

### 10.49.1 Description

TParams is a collection of TParam (395) values. It is used to specify parameter values for parametrized SQL statements, but is also used to specify parameter values for stored procedures. Its default property is an array of TParam (395) 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 (410).

TDataset (285) itself does not use TParams. The class is provided in the DB unit, so all TDataset descendants that need some kind of parametrization make use of the same interface. The TMasterParamsDataLink (390) 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 (411) method exists.

See also: TDataset (285), TMasterParamsDataLink (390), TParam (395), TParams.ParseSQL (410), TParams.CopyParamValuesFromDataset (411)

### 10.49.2 Method overview

Page	Property	Description
408	AddParam	Add a parameter to the collection
408	AssignValues	Copy values from another collection
411	CopyParamValuesFromDataset	Copy parameter values from the fields in a dataset.
408	Create	Create a new instance of TParams
408	CreateParam	Create and add a new parameter to the collection
409	FindParam	Find a parameter with given name
409	GetParamList	Fetch a list of TParam instances
409	IsEqual	Is the list of parameters equal
410	ParamByName	Return a parameter by name
410	ParseSQL	Parse SQL statement, replacing parameter names with SQL parameter placeholders
411	RemoveParam	Remove a parameter from the collection

### 10.49.3 Property overview

Page	Property	Access	Description
<a href="#">411</a>	Dataset	r	Dataset that owns the TParams instance
<a href="#">412</a>	Items	rw	Indexed access to TParams instances in the collection
<a href="#">412</a>	ParamValues	rw	Named access to the parameter values.

### 10.49.4 TParams.Create

**Synopsis:** Create a new instance of TParams

**Declaration:** 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 ([395](#)) 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 ([395](#))

### 10.49.5 TParams.AddParam

**Synopsis:** Add a parameter to the collection

**Declaration:** procedure AddParam(Value: TParam)

**Visibility:** public

**Description:** AddParam adds Value to the collection.

**Errors:** No checks are done on the TParam instance. If it is Nil, an exception will be raised.

**See also:** TParam ([395](#)), #rtl.classes.tcollection.add (??)

### 10.49.6 TParams.AssignValues

**Synopsis:** Copy values from another collection

**Declaration:** procedure AssignValues(Value: TParams)

**Visibility:** public

**Description:** AssignValues examines all TParam ([395](#)) 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 ([397](#))). If it is not found, nothing is done.

**See also:** TParam ([395](#)), TParam.Assign ([397](#))

### 10.49.7 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 (395) 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 (395), TParam.Name (405), TParam.Datatype (405), TParam.Paramtype (406)

### **10.49.8 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 (395) 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 (405), TParams.ParamByName (410), TParams.GetParamList (409)

### **10.49.9 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 (395), TParam.Name (405), TParams.ParamByName (410)

### **10.49.10 TParams.AreEqual**

Synopsis: Is the list of parameters equal

Declaration: function AreEqual(Value: TParams) : Boolean

Visibility: public

Description: `.AreEqual` 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 (395)

### 10.49.11 TParams.ParamByName

**Synopsis:** Return a parameter by name

**Declaration:** function ParamByName(const Value: string) : TParam

**Visibility:** public

**Description:** ParamByName searches the collection for the TParam (395) 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 (405), TParams.FindParam (409), TParams.GetParamList (409)

### 10.49.12 TParams.ParseSQL

**Synopsis:** Parse SQL statement, replacing parameter names with SQL parameter placeholders

**Declaration:** function ParseSQL(SQL: string;DoCreate: Boolean) : string; Overload  
                  function ParseSQL(SQL: string;DoCreate: Boolean;EscapeSlash: Boolean;  
                           EscapeRepeat: Boolean;ParameterStyle: TParamStyle)  
                           : string; Overload  
                  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

**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 \(395\)](#), [TParam.Name \(405\)](#), [TParamStyle \(242\)](#)

### 10.49.13 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 \(395\)](#)

### 10.49.14 TParams.CopyParamValuesFromDataset

**Synopsis:** Copy parameter values from a 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 (397)`. By default, this operation is only performed on `TParam` instances with their `Bound (403)` 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 \(395\)](#), [TParam.Bound \(403\)](#), [TParam.AssignField \(397\)](#), [TDataSet \(285\)](#), [TDataSet.FieldName \(296\)](#)

### 10.49.15 TParams.Dataset

**Synopsis:** Dataset that owns the `TParams` instance

**Declaration:** Property Dataset : TDataSet

**Visibility:** public

**Access:** Read

**Description:** `Dataset` is the `TDataSet (285)` instance that was specified when the `TParams` instance was created.

See also: [TParams.Create \(408\)](#), [TDataSet \(285\)](#)

### 10.49.16 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 ([407](#))

### 10.49.17 TParams.ParamValues

**Synopsis:** Named access to the parameter values.

**Declaration:** Property ParamValues[ParamName: string]: Variant

**Visibility:** public

**Access:** Read,Write

**Description:** ParamValues provides access to the parameter values (TParam.Value ([405](#))) by name. It is equivalent to reading and writing

```
ParamByName(ParamName).Value
```

**See also:** TParam.Value ([405](#)), TParams.ParamByName ([410](#))

## 10.50 TSmallintField

### 10.50.1 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 ([334](#)), TNumericField ([394](#)), TLongintField ([385](#)), TWordField ([419](#))

### 10.50.2 Method overview

Page	Property	Description
<a href="#">413</a>	Create	Create a new instance of the TSmallintField class.

### 10.50.3 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 (412) class. It calls the inherited constructor and then simply sets some of the TField (334) properties to work with 16-bit signed integer data.

See also: [TField \(334\)](#)

## 10.51 TStringField

### 10.51.1 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 (334) 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 \(334\)](#), [TWideStringField \(418\)](#), [TDataSet \(285\)](#)

### 10.51.2 Method overview

Page	Property	Description
<a href="#">413</a>	Create	Create a new instance of the TStringField class
<a href="#">414</a>	SetFieldType	Set the field type

### 10.51.3 Property overview

Page	Property	Access	Description
<a href="#">415</a>	EditMask		Specify an edit mask for an edit control
<a href="#">414</a>	FixedChar	rw	Is the string declared with a fixed length ?
<a href="#">415</a>	Size		Maximum size of the string
<a href="#">414</a>	Transliterate	rw	Should the field value be transliterated when reading or writing
<a href="#">414</a>	Value	rw	Value of the field as a string

### 10.51.4 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 (334) properties after having called the inherited constructor.

### 10.51.5 TStringField.SetFieldType

**Synopsis:** Set the field type

**Declaration:** procedure SetFieldType(AValue: TFieldType);   Override

**Visibility:** public

**Description:** SetFieldType is overridden in TStringField (413) to check the data type more accurately (ftString and ftFixedChar). No extra functionality is added.

**See also:** TField.DataType (346)

### 10.51.6 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.

### 10.51.7 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 (305).

**See also:** TDataset.Translate (305)

### 10.51.8 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 (343) when read, or sets the AsString property when written to.

**See also:** TField.AsString (343), TField.Value (350)

### **10.51.9 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 \(347\)](#)

### **10.51.10 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 \(349\)](#)

## **10.52 TTimeField**

### **10.52.1 Description**

TimeField is the class used when a dataset must manage data of type time. (TField.DataTypee (346) equals ftTime). It initializes some of the properties of the TField (334) 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 \(285\)](#), [TField \(334\)](#), [TDateTimeField \(326\)](#), [TDateField \(326\)](#)

### **10.52.2 Method overview**

Page	Property	Description
<a href="#">416</a>	Create	Create a new instance of a TTimeField class.

### 10.52.3 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 (334) properties to configure the instance for working with time values.

See also: TField (334)

## 10.53 TVarBytesField

### 10.53.1 Description

TVarBytesField is the class used when a dataset must manage data of variable-size binary type. (TField.DataType (346) equals ftVarBytes). It initializes some of the properties of the TField (334) class to be able to work with variable-size bytes.

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 (285), TField (334), TBytesField (266)

### 10.53.2 Method overview

Page	Property	Description
416	Create	Create a new instance of a TVarBytesField class.

### 10.53.3 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 (334) properties to configure the instance for working with variable-size binary data values.

See also: TField (334)

## 10.54 TVariantField

### 10.54.1 Description

TVariantField is the class used when a dataset must manage native variant-typed data. (TField.DataType (346) equals ftVariant). It initializes some of the properties of the TField (334) 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](#) (285), [TField](#) (334)

### 10.54.2 Method overview

Page	Property	Description
<a href="#">417</a>	Create	Create a new instance of the <a href="#">TVariantField</a> class

### 10.54.3 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](#) (334) properties to configure the instance for working with variant values.

See also: [TField](#) (334)

## 10.55 TWideMemoField

### 10.55.1 Description

[TWideMemoField](#) is the class used when a dataset must manage memo (Text BLOB) data. ([TField.DataType](#) (346) equals `ftWideMemo`). It initializes some of the properties of the [TField](#) (334) class. All methods to be able to work with widestring memo fields have been implemented in the [TBlobField](#) (261) 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](#) (285), [TField](#) (334), [TBlobField](#) (261), [TMemoField](#) (392), [TGraphicField](#) (376)

### 10.55.2 Method overview

Page	Property	Description
<a href="#">417</a>	Create	Create a new instance of the <a href="#">TWideMemoField</a> class

### 10.55.3 Property overview

Page	Property	Access	Description
<a href="#">418</a>	Value	rw	Value of the field's contents as a widestring

### 10.55.4 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 (334) properties to configure the instance for working with widestring memo values.

See also: TField (334)

### 10.55.5 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 (344) property.

See also: TField.Value (350), Tfield.AsWideString (344)

## 10.56 TWideStringField

### 10.56.1 Description

TWideStringField is the string field class instantiated for fields of data type ftWideString. This class overrides some of the standard TField (334) 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 (334), TStringField (413), TDataset (285)

### 10.56.2 Method overview

Page	Property	Description
418	Create	Create a new instance of the TWideStringField class.
419	SetFieldType	Set the field type

### 10.56.3 Property overview

Page	Property	Access	Description
419	Value	rw	Value of the field as a widestring

### 10.56.4 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 (334) properties after having called the inherited constructor.

### 10.56.5 TWideStringField.SetFieldType

**Synopsis:** Set the field type

**Declaration:** procedure SetFieldType(AValue: TFieldType); Override

**Visibility:** public

**Description:** SetFieldType is overridden in TWideStringField (418) to check the data type more accurately (ftWideString and ftFixedWideChar). No extra functionality is added.

**See also:** [TField.DataType \(346\)](#)

### 10.56.6 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 \(344\)](#) property.

**See also:** [TField.AsWideString \(344\)](#), [TField.Value \(350\)](#)

## 10.57 TWordField

### 10.57.1 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 \(334\)](#), [TNumericField \(394\)](#), [TLongintField \(385\)](#), [TSmallIntField \(412\)](#)

### 10.57.2 Method overview

Page	Property	Description
<a href="#">419</a>	Create	Create a new instance of the TWordField class.

### 10.57.3 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 (419) class. It calls the inherited constructor and then simply sets some of the [TField \(334\)](#) properties to work with 16-bit unsigned integer data.

**See also:** [TField \(334\)](#)

# Chapter 11

## Reference for unit 'dbugintf'

### 11.1 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 (`dbugsrv`) 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 (420) 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 (420) unit should also be functional.

### 11.2 Writing a debug server

Writing a debug server is relatively easy. It should instantiate a `TSimpleIPCServer` class from the SimpleIPC (420) 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 (420) 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 (420) unit should also be functional.

### 11.3 Constants, types and variables

#### 11.3.1 Resource strings

`SEntering = '> Entering '`

String used when sending method enter message.

`SExiting = '< Exiting '`

String used when sending method exit message.

```
SProcessID = 'Process %s'
```

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

### 11.3.2 Constants

```
SendError : string = ''
```

Whenever a call encounteres an exception, the exception message is stored in this variable.

### 11.3.3 Types

```
TDebugLevel = (dlInformation, dlWarning, dlError)
```

Table 11.1: Enumeration values for type TDebugLevel

Value	Explanation
dlError	Error message
dlInformation	Informational message
dlWarning	Warning message

TDebugLevel indicates the severity level of the debug message to be sent. By default, an informational message is sent.

## 11.4 Procedures and functions

### 11.4.1 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 (425), SendDebug (422)

### 11.4.2 InitDebugClient

**Synopsis:** Initialize the debug client.

**Declaration:** function InitDebugClient : 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 ([422](#)) call will initialize the debug client when it is first called.

**Errors:** None.

**See also:** SendDebug ([422](#)), StartDebugServer ([425](#))

### 11.4.3 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 ([422](#)) which sends the name and value of a boolean value as an informational message.

**Errors:** None.

**See also:** SendDebug ([422](#)), SendDateTime ([422](#)), SendInteger ([424](#)), SendPointer ([425](#))

### 11.4.4 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 ([422](#)) 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 ([422](#)), SendBoolean ([422](#)), SendInteger ([424](#)), SendPointer ([425](#))

### 11.4.5 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` ([421](#)).

**See also:** `SendDebugEx` ([423](#)), `SendDebugFmt` ([423](#)), `SendDebugFmtEx` ([423](#))

### 11.4.6 `SendDebugEx`

**Synopsis:** Send debug message other than informational messages

**Declaration:** `procedure SendDebugEx(const Msg: string; MType: TDebugLevel)`

**Visibility:** default

**Description:** `SendDebugEx` allows to specify the debug level of the message to be sent in `MType`. By default, `SendDebug` ([422](#)) uses informational messages.

Other than that the function of `SendDebugEx` is equal to that of `SendDebug`

**Errors:** None.

**See also:** `SendDebug` ([422](#)), `SendDebugFmt` ([423](#)), `SendDebugFmtEx` ([423](#))

### 11.4.7 `SendDebugFmt`

**Synopsis:** Format and send a debug message

**Declaration:** `procedure SendDebugFmt(const Msg: string; const Args: Array of const)`

**Visibility:** default

**Description:** `SendDebugFmt` is a utility routine which formats a message by passing `Msg` and `Args` to `Format` ([??](#)) and sends the result to the debug server using `SendDebug` ([422](#)). It exists mainly to avoid the `Format` call in calling code.

**Errors:** None.

**See also:** `SendDebug` ([422](#)), `SendDebugEx` ([423](#)), `SendDebugFmtEx` ([423](#)), `#rtl.sysutils.format` ([??](#))

### 11.4.8 `SendDebugFmtEx`

**Synopsis:** Format and send message with alternate type

**Declaration:** `procedure SendDebugFmtEx(const Msg: string; const Args: Array of const; MType: TDebugLevel)`

**Visibility:** default

**Description:** `SendDebugFmtEx` is a utility routine which formats a message by passing `Msg` and `Args` to `Format` ([??](#)) and sends the result to the debug server using `SendDebugEx` ([423](#)) with Debug level `MType`. It exists mainly to avoid the `Format` call in calling code.

**Errors:** None.

**See also:** `SendDebug` ([422](#)), `SendDebugEx` ([423](#)), `SendDebugFmt` ([423](#)), `#rtl.sysutils.format` ([??](#))

### 11.4.9 SendInteger

**Synopsis:** Send the value of an integer variable.

**Declaration:** `procedure SendInteger(const Identifier: string; const Value: Integer;  
HexNotation: Boolean)`

**Visibility:** default

**Description:** `SendInteger` is a simple wrapper around `SendDebug` (422) 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` (422), `SendBoolean` (422), `SendDateTime` (422), `SendPointer` (425)

### 11.4.10 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` (424), 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` (422), `SendMethodExit` (424), `SendSeparator` (425)

### 11.4.11 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` (424), 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` (422), `SendMethodEnter` (424), `SendSeparator` (425)

### 11.4.12 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 (422) 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 (422), SendBoolean (422), SendDateTime (422), SendInteger (424)

### 11.4.13 SendSeparator

**Synopsis:** Send a separator message

**Declaration:** procedure SendSeparator

**Visibility:** default

**Description:** SendSeparator is a simple wrapper around SendDebug (422) 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 (422), SendMethodEnter (424), SendMethodExit (424)

### 11.4.14 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 (422) 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 (421), SendDebug (422)

### 11.4.15 StartDebugServer

**Synopsis:** Start the debug server

**Declaration:** function StartDebugServer : 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` (422) call will attempt to start the server by itself if it is not yet running.

**Errors:** On error, `False` is returned.

**See also:** `SendDebug` (422), `InitDebugClient` (422)

# Chapter 12

## Reference for unit 'dbugmsg'

### 12.1 Used units

Table 12.1: Used units by unit 'dbugmsg'

Name	Page
Classes	??
System	??

### 12.2 Overview

dbugmsg is an auxiliary unit used in the dbugintf (420) unit. It defines the message protocol used between the debug unit and the debug server.

### 12.3 Constants, types and variables

#### 12.3.1 Constants

```
DebugServerID : string = '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.

### 12.3.2 Types

```
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` (429) and `ReadDebugMessageFromStream` (428) can be used to read or write the message from/to a stream.

## 12.4 Procedures and functions

### 12.4.1 DebugMessageName

**Synopsis:** Return the name of the debug message

**Declaration:** function `DebugMessageName (msgType: Integer) : string`

**Visibility:** default

**Description:** `DebugMessageName` returns the name of the message type. It can be used to examine the `MsgType` field of a `TDebugMessage` (428) 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.

### 12.4.2 ReadDebugMessageFromStream

**Synopsis:** Read a message from stream

**Declaration:** procedure `ReadDebugMessageFromStream (AStream: TStream;`  
`var Msg: TDebugMessage)`

**Visibility:** default

**Description:** `ReadDebugMessageFromStream` reads a `TDebugMessage` (428) 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` (428), `WriteDebugMessageToStream` (429)

### **12.4.3 WriteDebugMessageToStream**

**Synopsis:** Write a message to stream

**Declaration:** procedure WriteDebugMessageToStream(AStream: TStream;  
const Msg: TDebugMessage)

**Visibility:** default

**Description:** WriteDebugMessageFromStream writes a TDebugMessage ([428](#)) 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 ([428](#)), ReadDebugMessageFromStream ([428](#))

# Chapter 13

## Reference for unit 'eventlog'

### 13.1 Used units

Table 13.1: Used units by unit 'eventlog'

Name	Page
Classes	??
System	??
sysutils	??

### 13.2 Overview

The EventLog unit implements the TEventLog (432) 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.

### 13.3 Constants, types and variables

#### 13.3.1 Resource strings

```
SErrMsgFailed = 'Failed to log entry (Error: %s)'
```

Message used to format an error when an error exception is raised.

```
SLogCustom = 'Custom (%d)'
```

Custom message formatting string

```
SLogDebug = 'Debug'
```

Debug message name

```
S.LogError = 'Error'
```

Error message name

```
SLogInfo = 'Info'
```

Informational message name

```
SLogWarning = 'Warning'
```

Warning message name

### 13.3.2 Types

```
TLogCategoryEvent = procedure(Sender: TObject; var Code: Word) of object
```

`TLogCategoryEvent` is the event type for the `TEventLog.OnGetCustomCategory` ([439](#)) 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` ([439](#)) and `OnGetCustomEventID` ([439](#)) event handlers. It should return a OS system log code for the `etCustom` log event or event ID type in the `Code` parameter.

```
TLogType = (ltSystem, ltFile)
```

Table 13.2: Enumeration values for type `TLogType`

Value	Explanation
ltFile	Write to file
ltSystem	Use the system log

`TLogType` determines where the log messages are written. It is the type of the `TEventLog.LogType` ([436](#)) 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.

## 13.4 ELogError

### 13.4.1 Description

`ELogError` is the exception used in the `TEventLog` ([432](#)) component to indicate errors.

See also: `TEventLog` ([432](#))

## 13.5 TEventLog

### 13.5.1 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 (434) call, or the specialized Warning (435), Error (435), Info (436) or Debug (435) calls, which have the event type predefined.

See also: Log (434), Warning (435), Error (435), Info (436), Debug (435)

### 13.5.2 Method overview

Page	Property	Description
435	Debug	Log a debug message
432	Destroy	Clean up TEventLog instance
435	Error	Log an error message to
433	EventTypeToString	Create a string representation of an event type
436	Info	Log an informational message
434	Log	Log a message to the system log.
434	Pause	Pause the sending of log messages.
433	RegisterMessageFile	Register message file
434	Resume	Resume sending of log messages if sending was paused
434	UnRegisterMessageFile	Unregister the message file (needed on windows only)
435	Warning	Log a warning message.

### 13.5.3 Property overview

Page	Property	Access	Description
437	Active	rw	Activate the log mechanism
436	AppendContent	rw	Control whether output is appended to an existing file
438	CustomLogType	rw	Custom log type ID
437	DefaultEventType	rw	Default event type for the Log (434) call.
438	EventIDOffset	rw	Offset for event ID messages identifiers
437	FileName	rw	File name for log file
436	Identification	rw	Identification string for messages
436	LogType	rw	Log type
439	OnGetCustomCategory	rw	Event to retrieve custom message category
439	OnGetCustomEvent	rw	Event to retrieve custom event Code
439	OnGetCustomEventID	rw	Event to retrieve custom event ID
440	Paused	rw	Is the message sending paused ?
437	RaiseExceptionOnError	rw	Determines whether logging errors are reported or ignored
438	TimeStampFormat	rw	Format for the timestamp string

### 13.5.4 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 (437) property to False.

See also: Active (437)

### 13.5.5 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 (??)

### 13.5.6 TEventLog.RegisterMessageFile

**Synopsis:** Register message file

**Declaration:** function RegisterMessageFile(AFileName: string) : Boolean; Virtual

**Visibility:** public

**Description:** RegisterMessageFile is used on Windows to register the file AFileName containing the formatting strings for the system messages. This should be a file containing resource strings. If AFileName is empty, the filename of the application binary is substituted.

When a message is logged to the windows system log, Windows looks for a formatting string in the file registered with this call.

There are 2 kinds of formatting strings:

**Category strings** these should be numbered from 1 to 4

- 1Should contain the description of the etInfo event type.
- 2Should contain the description of the etWarning event type.
- 4Should contain the description of the etError event type.
- 4Should contain the description of the etDebug event type.

None of these strings should have a string substitution placeholder.

The second type of strings are the **message definitions**. Their number starts at EventIDOffset (438) (default is 1000) and each string should have 1 placeholder.

Free Pascal comes with a fclel.res resource file which contains default values for the 8 strings, in english. It can be linked in the application binary with the statement

```
{$R fclel.res}
```

This file is generated from the fclel.mc and fclel.rc files that are distributed with the Free Pascal sources.

If the strings are not registered, windows will still display the event messages, but they will not be formatted nicely.

Note that while any messages logged with the event logger are displayed in the event viewern Windows locks the file registered here. This usually means that the binary is locked.

On non-windows operating systems, this call is ignored.

**Errors:** If AFileName is invalid, false is returned.

### 13.5.7 TEventLog.UnRegisterMessageFile

**Synopsis:** Unregister the message file (needed on windows only)

**Declaration:** function UnRegisterMessageFile : Boolean; Virtual

**Visibility:** public

**Description:** UnRegisterMessageFile can be used to unregister a message file previously registered with TEventLog.RegisterMessageFile (433). This function is usable only on windows, it has no effect on other platforms. Note that windows locks the registered message file while viewing messages, so unregistering helps to avoid file locks while event viewer is open.

**See also:** TEventLog.RegisterMessageFile (433)

### 13.5.8 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 (430).

**See also:** TEventLog.Resume (434), TEventLog.Paused (440)

### 13.5.9 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 (430).

**See also:** TEventLog.Pause (434), TEventLog.Paused (440)

### 13.5.10 TEventLog.Log

**Synopsis:** Log a message to the system log.

**Declaration:** procedure Log(EventType: TEventType; const Msg: string); Overload  
procedure Log(EventType: TEventType; const Fmt: string;  
                 Args: Array of const); Overload  
procedure Log(const Msg: string); Overload  
procedure Log(const Fmt: string; Args: Array of const); Overload

**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` (437).

If `EventType` is `etCustom`, then the `OnGetCustomEvent` (439), `OnGetCustomEventID` (439) and `OnGetCustomCategory` (439).

The other logging calls: `Info` (436), `Warning` (435), `Error` (435) and `Debug` (435) use the `Log` call to do the actual work.

**See also:** `Info` (436), `Warning` (435), `Error` (435), `Debug` (435), `OnGetCustomEvent` (439), `OnGetCustomEventID` (439), `OnGetCustomCategory` (439)

### 13.5.11 TEventLog.Warning

**Synopsis:** Log a warning message.

**Declaration:** `procedure Warning(const Msg: string); Overload`  
`procedure Warning(const Fmt: string; Args: Array of const); Overload`

**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` (434), `Info` (436), `Error` (435), `Debug` (435)

### 13.5.12 TEventLog.Error

**Synopsis:** Log an error message to

**Declaration:** `procedure Error(const Msg: string); Overload`  
`procedure Error(const Fmt: string; Args: Array of const); Overload`

**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` (434), `Info` (436), `Warning` (435), `Debug` (435)

### 13.5.13 TEventLog.Debug

**Synopsis:** Log a debug message

**Declaration:** `procedure Debug(const Msg: string); Overload`  
`procedure Debug(const Fmt: string; Args: Array of const); Overload`

**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` (434), `Info` (436), `Warning` (435), `Error` (435)

### 13.5.14 TEventLog.Info

**Synopsis:** Log an informational message

**Declaration:** procedure Info(const Msg: string); Overload  
procedure Info(const Fmt: string; Args: Array of const); Overload

**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 (434), Warning (435), Error (435), Debug (435)

### 13.5.15 TEventLog.AppendContent

**Synopsis:** Control whether output is appended to an existing file

**Declaration:** Property AppendContent : Boolean

**Visibility:** published

**Access:** Read,Write

**Description:** AppendContent determines what is done when the log type is `ltFile` and a log file already exists. If the log file already exists, then the default behaviour (`AppendContent=False`) is to re-create the log file when the log is activated. If `AppendContent` is True then output will be appended to the existing file.

**See also:** LogType (436), FileName (437)

### 13.5.16 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 (437), TimeStampFormat (438)

### 13.5.17 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 (437) property.

**See also:** FileName (437)

### 13.5.18 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 ([434](#))

### 13.5.19 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.

### 13.5.20 TEventLog.DefaultEventType

**Synopsis:** Default event type for the Log ([434](#)) call.

**Declaration:** Property DefaultEventType : TEventType

**Visibility:** published

**Access:** Read,Write

**Description:** DefaultEventType is the event type used by the Log ([434](#)) call if it's EventType parameter is omitted.

**See also:** Log ([434](#))

### 13.5.21 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` (436) 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` (436)

### 13.5.22 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` (436)

### 13.5.23 TEventLog.CustomLogType

**Synopsis:** Custom log type ID

**Declaration:** `Property CustomLogType : Word`

**Visibility:** published

**Access:** Read,Write

**Description:** `CustomLogType` is used in the `EventTypeToString` (433) to format the custom log event type string.

See also: `EventTypeToString` (433)

### 13.5.24 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` (433) 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 ([433](#))

### **13.5.25 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 ([439](#)), OnGetCustomEvent ([439](#))

### **13.5.26 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 ([439](#)), OnGetCustomEvent ([439](#))

### **13.5.27 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 ([439](#)), OnGetCustomEventID ([439](#))

### **13.5.28 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 ([430](#)), setting it to False has the same effect as calling Resume ([430](#)).

**See also:** TEventLog.Pause ([434](#)), TEventLog.Resume ([434](#))

# Chapter 14

## Reference for unit 'ezcgi'

### 14.1 Used units

Table 14.1: Used units by unit 'ezcgi'

Name	Page
Classes	??
System	??
sysutils	??

### 14.2 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 ([444](#)) or DoGet ([444](#)) methods should be overridden.

### 14.3 Constants, types and variables

#### 14.3.1 Constants

```
hexTable = '0123456789ABCDEF'
```

String constant used to convert a number to a hexadecimal code or back.

### 14.4 ECGIException

#### 14.4.1 Description

Exception raised by TEZcgi ([442](#))

See also: TEZcgi (442)

## 14.5 TEZcgi

### 14.5.1 Description

TEZcgi implements all functionality to analyze the CGI environment and query the variables present in it. Its main use is the exposed variables.

Programs wishing to use this class should make a descendent class of this class and override the DoPost (444) or DoGet (444) methods. To run the program, an instance of this class must be created, and its Run (443) 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.

### 14.5.2 Method overview

Page	Property	Description
<a href="#">442</a>	Create	Creates a new instance of the TEZCGI component
<a href="#">442</a>	Destroy	Removes the TEZCGI component from memory
<a href="#">444</a>	DoGet	Method to handle GET requests
<a href="#">444</a>	DoPost	Method to handle POST requests
<a href="#">444</a>	GetValue	Return the value of a request variable.
<a href="#">443</a>	PutLine	Send a line of output to the web-client
<a href="#">443</a>	Run	Run the CGI application.
<a href="#">443</a>	WriteContent	Writes the content type to standard output

### 14.5.3 Property overview

Page	Property	Access	Description
<a href="#">446</a>	Email	rw	Email of the server administrator
<a href="#">446</a>	Name	rw	Name of the server administrator
<a href="#">445</a>	Names	r	Indexed array with available variable names.
<a href="#">444</a>	Values	r	Variables passed to the CGI script
<a href="#">446</a>	VariableCount	r	Number of available variables.
<a href="#">445</a>	Variables	r	Indexed array with variables as name=value pairs.

### 14.5.4 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 (445) property.

See also: Variables (445), Names (445), Values (444)

### 14.5.5 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 \(442\)](#)

### **14.5.6 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` (445) property) and calls the `DoPost` (444) or `DoGet` (444) 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 \(445\)](#), [DoPost \(444\)](#), [DoGet \(444\)](#)

### **14.5.7 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` (444) or `DoGet` (444) methods.

**See also:** [DoPost \(444\)](#), [DoGet \(444\)](#), [PutLine \(443\)](#)

### **14.5.8 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` (443) 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 \(443\)](#)

### 14.5.9 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 (444)

### 14.5.10 TEZcgi.DoPost

**Synopsis:** Method to handle POST requests

**Declaration:** procedure DoPost; Virtual

**Visibility:** public

**Description:** DoPost is called by the Run (443) method the POST method was used to invoke the CGI application. It should be overridden in descendants of TEZcgi to actually handle the request.

**See also:** Run (443), DoGet (444)

### 14.5.11 TEZcgi.DoGet

**Synopsis:** Method to handle GET requests

**Declaration:** procedure DoGet; Virtual

**Visibility:** public

**Description:** DoGet is called by the Run (443) method the GET method was used to invoke the CGI application. It should be overridden in descendants of TEZcgi to actually handle the request.

**See also:** Run (443), DoPost (444)

### 14.5.12 TEZcgi.Values

**Synopsis:** Variables passed to the CGI script

**Declaration:** Property Values[Index: string]: string

**Visibility:** public

**Access:** Read

**Description:** Values is a name-based array of variables that were passed to the script by the web server or the HTTP request. The Index variable is the name of the variable whose value should be retrieved. The following standard values are available:

**AUTH\_TYPE**Authorization type

**CONTENT\_LENGTH**Content length

**CONTENT\_TYPE**Content type

---

**GATEWAY\_INTERFACE**Used gateway interface  
**PATH\_INFO**Requested URL  
**PATH\_TRANSLATED**Transformed URL  
**QUERY\_STRING**Client query string  
**REMOTE\_ADDR**Address of remote client  
**REMOTE\_HOST**DNS name of remote client  
**REMOTE\_IDENT**Remote identity.  
**REMOTE\_USER**Remote user  
**REQUEST\_METHOD**Request methods (POST or GET)  
**SCRIPT\_NAME**Script name  
**SERVER\_NAME**Server host name  
**SERVER\_PORT**Server port  
**SERVER\_PROTOCOL**Server protocol  
**SERVER\_SOFTWARE**Web server software  
**HTTP\_ACCEPT**Accepted responses  
**HTTP\_ACCEPT\_CHARSET**Accepted character sets  
**HTTP\_ACCEPT\_ENCODING**Accepted encodings  
**HTTP\_IF\_MODIFIED\_SINCE**Proxy information  
**HTTP\_REFERER**Referring page  
**HTTP\_USER\_AGENT**Client software name

Other than the standard list, any variables that were passed by the web-client request, are also available. Note that the variables are case insensitive.

See also: [TEZCGI.Variables \(445\)](#), [TEZCGI.Names \(445\)](#), [TEZCGI.GetValue \(444\)](#), [TEZcgi.VariableCount \(446\)](#)

### 14.5.13 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 ([446](#)). Any other value will result in an exception being raised.

See also: [TEZcgi.Variables \(445\)](#), [TEZcgi.Values \(444\)](#), [TEZcgi.GetValue \(444\)](#), [TEZcgi.VariableCount \(446\)](#)

### 14.5.14 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 ([446](#)). Any other value will result in an exception being raised.

**See also:** TEZcgi.Names ([445](#)), TEZcgi.Values ([444](#)), TEZcgi.GetValue ([444](#)), TEZcgi.VariableCount ([446](#))

### **14.5.15 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 ([445](#)) and Variables ([445](#)) properties.

**See also:** Names ([445](#)), Variables ([445](#)), TEZcgi.Values ([444](#)), TEZcgi.GetValue ([444](#))

### **14.5.16 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 ([443](#)) method.

**See also:** TEZcgi.Run ([443](#)), TEZcgi.Email ([446](#))

### **14.5.17 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 ([443](#)) method.

**See also:** TEZcgi.Run ([443](#)), TEZcgi.Name ([446](#))

# Chapter 15

## Reference for unit 'fpjson'

### 15.1 Used units

Table 15.1: Used units by unit 'fpjson'

Name	Page
Classes	??
contnrs	<a href="#">119</a>
System	??
sysutils	??
variants	??

### 15.2 Overview

The JSON unit implements JSON support for Free Pascal. It contains the data structures (`TJSONData` ([467](#)) and descendent objects) to treat JSON data and output JSON as a string `TJSONData.AsJSON` ([475](#)). The generated JSON can be formatted in several ways `TJSONData.FormatJSON` ([471](#)).

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` ([477](#)), `TJSONFloatNumber` ([475](#)) or `TJSONInt64Number` ([476](#)), depending on the type of the number.

**Strings** in `TJSONString` ([489](#)).

**Boolean** in `TJSONBoolean` ([465](#)).

**null** is supported using `TJSONNull` ([478](#))

**Array** is supported using `TJSONArray` ([457](#))

**Object** is supported using `TJSONObject` ([480](#))

The constructors of these objects allow to set the value, making them very easy to use. The memory management is automatic in the sense that arrays and objects own their values, and when the array or object is freed, all data in it is freed as well.

Typical use would be:

```

Var
  O : TJSONObject;

begin
  O:=TJSONObject.Create(['Age', 44,
                        'Firstname','Michael',
                        'Lastname','Van Canneyt']);
  Writeln(O.AsJSON);
  Write('Welcome ',O.Strings['Firstname'],', ');
  Writeln(O.Get('Lastname','')); // empty default.
  Writeln(' , your current age is ',O.Integers('Age'));
  O.Free;
end;

```

The `TJSONArray` and `TJSONObject` classes offer methods to examine, get and set the various members and search through the various members.

Currently the JSON support only allows the use of UTF-8 data.

Parsing incoming JSON and constructing the JSON data structures is not implemented in the `fpJSON` unit. For this, the `jsonscanner` (447) unit must be included in the program unit clause. This sets several callback hooks (using `SetJSONParserHandler` (455) and then the `GetJSON` (453) 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);
end.

```

will print "Maria".

The FPJSON code does not use hardcoded class names when creating the JSON: it uses the various `CreateJSON` (452) functions to create the data. These functions use a registry of classes, so it is possible to create descendants of the classes in the `fpjson` unit and have these used for construction of JSON Data structures. The `GetJSONInstanceType` (454) and `SetJSONInstanceType` (455) functions can be used to get or set the classes that must be used. the default parser used by `GetJSON` (453) will also use these functions.

## 15.3 Constants, types and variables

### 15.3.1 Constants

```
AsJSONFormat = [foSingleLineArray, foSingleLineObject]
```

AsJSONFormat contains the options that make TJSONData.FormatJSON (471) behave like TJSONData.AsJSON (475)

```
DefaultFormat = []
```

DefaultFormat contains the default formatting options used in formatted JSON.

```
DefaultIndentSize = 2
```

DefaultIndentSize is the default indent size used in formatted JSON.

### 15.3.2 Types

```
PJSONCharType = ^TJSONCharType
```

PJSONCharType is a pointer to a TJSONCharType (450) character. It is used while parsing JSON.

```
TFormatOption = (foSingleLineArray, foSingleLineObject,
                 foDoNotQuoteMembers, foUseTabchar)
```

Table 15.2: Enumeration values for type TFormatOption

Value	Explanation
foDoNotQuoteMembers	Do not use quote characters around object member names.
foSingleLineArray	Keep all array elements on a single line.
foSingleLineObject	Keep all object elements on a single line.
foUseTabchar	Use the tabulator character for indents

TFormatOption enumerates the various formatting options that can be used in the TJSONData.FormatJSON (471) function.

```
TFormatOptions = Set of TFormatOption
```

TFormatOptions is the set definition used to specify options in TJSONData.FormatJSON (471).

```
TJSONArrayClass = Class of TJSONArray
```

```
TJSONArrayIterator = procedure(Item: TJSONData; Data: TObject;
                                var Continue: Boolean) of object
```

TJSONArrayIterator is the procedural callback used by TJSONArray.Iterate (459) 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 ([465](#)). It is used in the factory methods.

```
TJSONCharType = AnsiChar
```

TJSONCharType is the type of a single character in a TJSONStringType ([452](#)) string. It is used by the parser.

```
TJSONDataClass = Class of TJSONData
```

TJSONDataClass is used in the CreateJSON ([452](#)), SetJSONInstanceType ([455](#)) and GetJSONInstanceType ([454](#)) functions to set the actual classes used when creating JSON data.

```
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).

```
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 ([475](#)). It is used in the factory methods.

```
TJSONInstanceType = (jitUnknown, jitNumberInteger, jitNumberInt64,
                     jitNumberFloat, jitString, jitBoolean, jitNull,
                     jitArray, jitObject)
```

Table 15.3: Enumeration values for type TJSONInstanceType

Value	Explanation
jitArray	Array value
jitBoolean	Boolean value
jitNull	Null value
jitNumberFloat	Floating point real number value
jitNumberInt64	64-bit signed integer number value
jitNumberInteger	32-bit signed integer number value
jitObject	Object value
jitString	String value
jitUnknown	Unknown

TJSONInstanceType is used by the parser to determine what kind of TJSONData (467) descendant to create for a particular data item. It is a more fine-grained division than TJSontype (452)

```
TJSONInt64NumberClass = Class of TJSONInt64Number
```

TJSONInt64NumberClass is the class type of TJSONInt64Number (476). It is used in the factory methods.

```
TJSONIntegerNumberClass = Class of TJSONIntegerNumber
```

TJSONIntegerNumberClass is the class type of TJSONIntegerNumber (477). It is used in the factory methods.

```
TJSONNullClass = Class of TJSONNull
```

TJSONNullClass is the class type of TJSONNull (478). It is used in the factory methods.

```
TJSONNumberType = (ntFloat, ntInteger, ntInt64)
```

Table 15.4: Enumeration values for type TJSONNumberType

Value	Explanation
ntFloat	Floating point value
ntInt64	64-bit integer value
ntInteger	32-bit Integer value

TJSONNumberType is used to enumerate the different kind of numerical types: JSON only has a single 'number' format. Depending on how the value was parsed, FPC tries to create a value that is as close to the original value as possible: this can be one of integer, int64 or TJSONFloatType (normally a double). The number types have a common ancestor, and they are distinguished by their TJSONNumber.NumberType (480) value.

```
TJSONObjectClass = Class of TJSONObject
```

```
TJSONObjectIterator = procedure (const AName: TJSONStringType;
                                Item: TJSONData; Data: TObject;
                                var Continue: Boolean) of object
```

TJSONObjectIterator is the procedural callback used by TJSONObject.Iterate (483) 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 (453) 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 (455) method, the actual types can be retrieved with GetJSONInstanceType (454)

---

```
TJSONObjectClass = Class of TJSONObject
```

`TJSONObjectClass` is the class type of `TJSONObject` (489). It is used in the factory methods.

```
TJSONObjectType = AnsiString
```

`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.

```
TJSONObjectType = (jtUnknown, jtNumber, jtString, jtBoolean, jtNull, jtArray,
                    jtObject)
```

Table 15.5: Enumeration values for type `TJSONObjectType`

Value	Explanation
<code>jtArray</code>	Array data (integer index, elements can be any type)
<code>jtBoolean</code>	Boolean data
<code>jtNull</code>	Null data
<code>jtNumber</code>	Numerical type. This can be integer (32/64 bit) or float.
<code>jtObject</code>	Object data (named index, elements can be any type)
<code>jtString</code>	String data type.
<code>jtUnknown</code>	Unknown JSON data type

`TJSONObjectType` determines the type of JSON data a particular object contains. The class function `TJSONData.JSONType` (468) returns this type, and indicates what kind of data that particular descendent contains. The values correspond to the original data types in the JSON specification. The `TJSONData` object itself returns the unknown value.

## 15.4 Procedures and functions

### 15.4.1 CreateJSON

**Synopsis:** Create a JSON data item

**Declaration:**

```
function CreateJSON : TJSONObject
function CreateJSON(Data: Boolean) : TJSONObjectBoolean
function CreateJSON(Data: Integer) : TJSONObjectIntegerNumber
function CreateJSON(Data: Int64) : TJSONObjectInt64Number
function CreateJSON(Data: TJSONFloat) : TJSONFloatNumber
function CreateJSON(Data: TJSONObjectType) : TJSONObject
```

**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` (454) to do so. The classes to be used can be set using the `SetJSONInstanceType` (455).

The JSON parser uses these functions to create instances of `TJSONData` (467).

**Errors:** None.

**See also:** `GetJSONInstanceType` (454), `SetJSONInstanceType` (455), `GetJSON` (453), `CreateJSONArray` (453), `CreateJSONObject` (453)

### 15.4.2 CreateJSONArray

**Synopsis:** Create a JSON array

**Declaration:** function CreateJSONArray(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 (454), SetJSONInstanceType (455), GetJSON (453), CreateJSON (452), TJSONArray (457)

### 15.4.3 CreateJSONObject

**Synopsis:** Create a JSON object

**Declaration:** function CreateJSONObject(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 (481) 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 (454), SetJSONInstanceType (455), GetJSON (453), CreateJSON (452), TJSONObject (480)

### 15.4.4 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 (455) 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 (454), SetJSONParserHandler (455), TJSONData (467)

### **15.4.5 GetJSONInstanceType**

**Synopsis:** JSON factory: Get the TJSONData class types to use

**Declaration:** function GetJSONInstanceType(AType: TJSONInstanceType) : TJSONDataClass

**Visibility:** default

**Description:** GetJSONInstanceType can be used to retrieve the registered descendants of the TJSONData (467) 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 (452) 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 (452), TJSONData (467), GetJSON (453)

### **15.4.6 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:** GetJSONParserHandler (454), TJSONParserHandler (451), GetJSON (453)

### **15.4.7 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:

\\" \\" \\\ \\\t \\\n \\\f \\\r \\\u000X

**See also:** StringToStringJSONString (455), JSONTypeName (454)

### **15.4.8 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:** StringToStringJSONString (455), JSONStringToString (454)

### **15.4.9 SetJSONInstanceType**

**Synopsis:** JSON factory: Set the JSONData class types to use

**Declaration:** procedure SetJSONInstanceType(AType: TJSONInstanceType;  
AClass: TJSONDataClass)

**Visibility:** default

**Description:** SetJSONInstanceType can be used to register descendants of the TJSONData ([467](#)) class, one for each possible kind of data. The class type used to instantiate data of type AType is passed in AClass.

The JSON parser will use the registered types to instantiate JSON Data instanced: when the parser encounters a value of type AType, it will instantiate a class of type AClass. By default, the classes in the fpJSON unit are used.

The CreateJSON ([452](#)) functions also use the types registered here to instantiate their data.

**Errors:** If AClass is not suitable to contain data of type AType, an exception is raised.

**See also:** GetJSONInstanceType ([454](#)), CreateJSON ([452](#))

### **15.4.10 SetJSONParserHandler**

**Synopsis:** Set the JSON parser handler

**Declaration:** procedure SetJSONParserHandler(AHandler: 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.

**See also:** GetJSONParserHandler ([454](#)), TJSONParserHandler ([451](#)), GetJSON ([453](#))

### **15.4.11 StringToJSONString**

**Synopsis:** Convert a string to a JSON-escaped string

**Declaration:** function StringToJSONString(const S: TJSONStringType) : 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.

**See also:** JSONStringToString ([454](#)), JSONTypeName ([454](#))

## 15.5 EJSON

### 15.5.1 Description

EJSON is the exception raised by the JSON implementation to report JSON error.

## 15.6 TBaseJSONEnumerator

### 15.6.1 Description

TBaseJSONEnumerator is the base type for the JSON enumerators. It should not be used directly, instead use the enumerator support of Object pascal to loop over values in JSON data.

The value of the TBaseJSONEnumerator enumerator is a record that describes the key and value of a JSON value. The key can be string-based (for records) or numerical (for arrays).

See also: [TJSONEnum \(450\)](#)

### 15.6.2 Method overview

Page	Property	Description
<a href="#">456</a>	GetCurrent	Return the current value of the enumerator
<a href="#">456</a>	MoveNext	Move to next value in array/object

### 15.6.3 Property overview

Page	Property	Access	Description
<a href="#">457</a>	Current	r	Return the current value of the enumerator

### 15.6.4 TBaseJSONEnumerator.GetCurrent

Synopsis: Return the current value of the enumerator

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

Visibility: public

Description: `GetCurrent` returns the current value of the enumerator. This is a [TJSONEnum \(450\)](#) value.

See also: [TJSONEnum \(450\)](#)

### 15.6.5 TBaseJSONEnumerator.MoveNext

Synopsis: Move to next value in array/object

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

Visibility: public

Description: `MoveNext` attempts to move to the next value. This will return `True` if the move was successful, or `False` if not. When `True` is returned, then

See also: [TJSONEnum \(450\)](#), [TJSONData \(467\)](#)

### 15.6.6 TBaseJSONEnumerator.Current

**Synopsis:** Return the current value of the enumerator

**Declaration:** Property Current : TJSEnum

**Visibility:** public

**Access:** Read

**Description:** Current returns the current enumerator value of type TJSEnum ([450](#)). It is only valid after MoveNext ([447](#)) returned True.

**See also:** TJSEnum ([450](#)), TJSDData ([467](#))

## 15.7 TJSONArray

### 15.7.1 Description

TJSONArrayClass is the class type of TJSONArray ([457](#)). It is used in the factory methods.

**See also:** TJSONArray ([457](#)), SetJSONInstanceType ([455](#)), GetJSONInstanceType ([454](#))

### 15.7.2 Method overview

Page	Property	Description
<a href="#">460</a>	Add	Add a JSON value to the array
<a href="#">459</a>	Clear	Clear the array
<a href="#">458</a>	Clone	Clone the JSON array
<a href="#">458</a>	Create	Create a new instance of JSON array data.
<a href="#">460</a>	Delete	Delete an element from the list by index
<a href="#">458</a>	Destroy	Free the JSON array
<a href="#">460</a>	Exchange	Exchange 2 elements in the list
<a href="#">461</a>	Extract	Extract an element from the array
<a href="#">459</a>	GetEnumerator	Get an array enumerator
<a href="#">459</a>	IndexOf	Return index of JSONData instance in array
<a href="#">461</a>	Insert	Insert an element in the array.
<a href="#">459</a>	Iterate	Iterate over all elements in the array
<a href="#">458</a>	JSONType	native JSON data type
<a href="#">461</a>	Move	Move a value from one location to another
<a href="#">462</a>	Remove	Remove an element from the list

### 15.7.3 Property overview

Page	Property	Access	Description
<a href="#">465</a>	Arrays	rw	Get or set elements as JSON array values
<a href="#">464</a>	Booleans	rw	Get or set elements as boolean values
<a href="#">464</a>	Floats	rw	Get or set elements as floating-point numerical values
<a href="#">463</a>	Int64s	rw	Get or set elements as Int64 values
<a href="#">463</a>	Integers	rw	Get or set elements as integer values
<a href="#">462</a>	Items		Indexed access to the values in the array
<a href="#">462</a>	Nulls	r	Check which elements are null
<a href="#">465</a>	Objects	rw	Get or set elements as JSON object values
<a href="#">464</a>	Strings	rw	Get or set elements as string values
<a href="#">462</a>	Types	r	JSON types of elements in the array

### **15.7.4 TJSONArray.Create**

**Synopsis:** Create a new instance of JSON array data.

**Declaration:** constructor Create; Overload; Reintroduce  
constructor Create(const Elements: Array of const); Overload

**Visibility:** public

**Description:** Create creates a new JSON array instance, and initializes the data with Elements. The elements are converted to various TJJSONData (467) instances, instances of TJJSONData 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 TJJSONNull 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 (454)

### **15.7.5 TJSONArray.Destroy**

**Synopsis:** Free the JSON array

**Declaration:** destructor Destroy; Override

**Visibility:** public

**Description:** Destroy will delete all elements in the array and clean up the TJSONArray (457) instance.

**See also:** TJSONArray.Clear (459), TJSONArray.Create (458)

### **15.7.6 TJSONArray.JSONType**

**Synopsis:** native JSON data type

**Declaration:** class function JSONType; Override

**Visibility:** public

**Description:** JSONType is overridden by TJSONArray to return jtArray.

**See also:** TJJSONData.JSONType (468)

### **15.7.7 TJSONArray.Clone**

**Synopsis:** Clone the JSON array

**Declaration:** function Clone : TJJSONData; 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:** TJJSONData.Clone (471)

### **15.7.8 TJSONArray.Iterate**

**Synopsis:** Iterate over all elements in the array

**Declaration:** procedure `Iterate(Iterator: TJSONArrayIterator; Data: TObject)`

**Visibility:** public

**Description:** `Iterate` iterates over all elements in the array, passing them one by one to the `Iterator` callback, together with the `Data` parameter. The iteration stops when all elements have been passed or when the iterator callback returned `False` in the `Continue` parameter.

**See also:** `TJSONArrayIterator` (449)

### **15.7.9 TJSONArray.IndexOf**

**Synopsis:** Return index of `TJSONData` instance in array

**Declaration:** function `IndexOf(obj: TJSONData) : Integer`

**Visibility:** public

**Description:** `IndexOf` compares all elements in the array with `Obj` and returns the index of the element instance that equals `Obj`. The actual instances are compared, not the JSON value. If none of the elements match, the function returns -1.

**See also:** `Clear` (447)

### **15.7.10 TJSONArray.GetEnumerator**

**Synopsis:** Get an array enumerator

**Declaration:** function `GetEnumerator : TBaseJSONEnumerator`; `Override`

**Visibility:** public

**Description:** `GetEnumerator` is overridden in `TJSONArray` so it returns an array enumerator. The array enumerator will return all the elements in the array, and stores their index in the `KeyNum` member of `TJSONEnum` (450).

**See also:** `TJSONEnum` (450), `TJSONData.GetEnumerator` (468)

### **15.7.11 TJSONArray.Clear**

**Synopsis:** Clear the array

**Declaration:** procedure `Clear;` `Override`

**Visibility:** public

**Description:** `Clear` clears the array and frees all elements in it. After the call to clear, `Count` (472) returns 0.

**See also:** `Delete` (447), `Extract` (447)

### **15.7.12 TJSONArray.Add**

**Synopsis:** Add a JSON value to the array

**Declaration:** function Add(Item: TJSONData) : Integer  
function Add(I: Integer) : Integer  
function Add(I: Int64) : Int64  
function Add(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 (467) 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 if it is an JSON array or object).

The function returns the TJSONData instance that was added to the array.

**See also:** Delete (447), Extract (447)

### **15.7.13 TJSONArray.Delete**

**Synopsis:** Delete an element from the list by index

**Declaration:** procedure Delete(Index: Integer)

**Visibility:** public

**Description:** Delete deletes the element with given Index from the list. The TJSONData (467) element is freed.

**Errors:** If an invalid index is passed, an exception is raised.

**See also:** Clear (447), Add (447), Extract (447), Exchange (447)

### **15.7.14 TJSONArray.Exchange**

**Synopsis:** Exchange 2 elements in the list

**Declaration:** procedure Exchange(Index1: Integer; Index2: Integer)

**Visibility:** public

**Description:** Exchange exchanges 2 elements at locations Index1 and Index2 in the list. This is more efficient than manually extracting and adding the elements to the list.

**Errors:** If an invalid index (for either element) is passed, an exception is raised.

### 15.7.15 TJSONArray.Extract

**Synopsis:** Extract an element from the array

**Declaration:** function Extract(Item: TJSONData) : TJSONData  
function Extract(Index: Integer) : TJSONData

**Visibility:** public

**Description:** Extract removes the element at position Index or the indicated element from the list, just as Delete ([447](#)) does. In difference with Delete, it does not free the object instance. Instead, it returns the extracted element.

**See also:** Delete ([447](#)), Clear ([447](#)), Insert ([447](#)), Add ([447](#))

### 15.7.16 TJSONArray.Insert

**Synopsis:** Insert an element in the array.

**Declaration:** procedure Insert(Index: Integer)  
procedure Insert(Index: Integer;Item: TJSONData)  
procedure Insert(Index: Integer;I: Integer)  
procedure Insert(Index: Integer;I: Int64)  
procedure Insert(Index: Integer;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 ([447](#)), it converts plain pascal values to JSON values.

Note that when inserting a TJSONData ([467](#)) 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 ([447](#)), Delete ([447](#)), Extract ([447](#)), Clear ([447](#))

### 15.7.17 TJSONArray.Move

**Synopsis:** Move a value from one location to another

**Declaration:** procedure Move(CurIndex: Integer;NewIndex: Integer)

**Visibility:** public

**Description:** Move moves the element at index CurIndex to the position NewIndex. It will shift the elements in between as needed. This operation is more efficient than extracting and inserting the element manually.

**See also:** Exchange ([447](#))

### **15.7.18 TJSONArray.Remove**

**Synopsis:** Remove an element from the list

**Declaration:** procedure Remove (Item: TJSONData)

**Visibility:** public

**Description:** Remove removes item from the array, if it is in the array. The object pointer is checked for presence in the array, not the JSON values. Note that the element is freed if it was in the array and is removed.

**See also:** Delete (447), Extract (447)

### **15.7.19 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 (472). TJSONArray simply declares it as the default property.

**See also:** TJSONData.Items (472)

### **15.7.20 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 (468) result of the elements in the array. Accessing it is equivalent to accessing

Items[Index].JSONType

**See also:** TJSONData.JSONType (468), TJSONData.Items (472)

### **15.7.21 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 (474) property when reading. It is then equivalent to accessing

Items[Index].IsNull

See also: [TJSONData.JSONType](#) (468), [TJSONData.Items](#) (472), [TJSONData.IsNull](#) (474), [TJSONArray.Types](#) (462)

### 15.7.22 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` (473) 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](#) (472), [TJSONData.IsNull](#) (474), [TJSONArray.Types](#) (462), [TJSONArray.Int64s](#) (463), [TJSONArray.Floats](#) (464), [TJSONArray.Strings](#) (464), [TJSONArrayBOOLEANS](#) (464)

### 15.7.23 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` (474) 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](#) (472), [TJSONData.IsNull](#) (474), [TJSONArray.Types](#) (462), [TJSONArray.Integers](#) (463), [TJSONArray.Floats](#) (464), [TJSONArray.Strings](#) (464), [TJSONArrayBOOLEANS](#) (464)

### **15.7.24 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 (472) 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 (472), TJSONData.IsNull (474), TJSONArray.Types (462), TJSONArray.Integers (463), TJSONArray.Floats (464), TJSONArray.Int64s (463), TJSONArrayBOOLEANS (464)

### **15.7.25 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 (473) 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 (472), TJSONData.IsNull (474), TJSONArray.Types (462), TJSONArray.Integers (463), TJSONArray.Strings (464), TJSONArray.Int64s (463), TJSONArrayBOOLEANS (464)

### **15.7.26 TJSONArrayBOOLEANS**

**Synopsis:** Get or set elements as boolean values

**Declaration:** PropertyBOOLEANS[Index: Integer]: Boolean

**Visibility:** public

**Access:** Read,Write

**Description:** Floats gives direct access to the TJSONData.AsBoolean (474) 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 \(472\)](#), [TJSONData.IsNull \(474\)](#), [TJSONArray.Types \(462\)](#), [TJSONArray.Integers \(463\)](#), [TJSONArray.Strings \(464\)](#), [TJSONArray.Int64s \(463\)](#), [TJSONArray.Floats \(464\)](#)

### 15.7.27 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 \(472\)](#), [TJSONData.IsNull \(474\)](#), [TJSONArray.Types \(462\)](#), [TJSONArray.Integers \(463\)](#), [TJSONArray.Strings \(464\)](#), [TJSONArray.Int64s \(463\)](#), [TJSONArray.Floats \(464\)](#), [TJSONArray.Objects \(465\)](#)

### 15.7.28 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 written value. Note that the old value is freed, and the new value is owned by the array.

See also: [TJSONData.Items \(472\)](#), [TJSONData.IsNull \(474\)](#), [TJSONArray.Types \(462\)](#), [TJSONArray.Integers \(463\)](#), [TJSONArray.Strings \(464\)](#), [TJSONArray.Int64s \(463\)](#), [TJSONArray.Floats \(464\)](#), [TJSONArray.Arrays \(465\)](#)

## 15.8 TJSONBoolean

### 15.8.1 Description

TJSONBoolean must be used whenever boolean data must be represented. It has limited functionality to convert the value from or to integer or floating point data.

See also: [TJSONFloatNumber \(475\)](#), [TJSONIntegerNumber \(477\)](#), [TJSONInt64Number \(476\)](#), [TJSONBoolean \(465\)](#), [TJSONNull \(478\)](#), [TJSONArray \(457\)](#), [TJSONObject \(480\)](#)

### 15.8.2 Method overview

Page	Property	Description
<a href="#">466</a>	Clear	Clear data
<a href="#">466</a>	Clone	Clone boolean value
<a href="#">466</a>	Create	Create a new instance of boolean JSON data
<a href="#">466</a>	JSONType	native JSON data type

### 15.8.3 TJSONBoolean.Create

Synopsis: Create a new instance of boolean JSON data

Declaration: constructor Create(AValue: Boolean); Reintroduce

Visibility: public

Description: Create instantiates a new boolean JSON data and initializes the value with AValue.

See also: TJSONIntegerNumber.Create ([478](#)), TJSONFloatNumber.Create ([475](#)), TJSONInt64Number.Create ([476](#)), TJSONString.Create ([490](#)), TJSONNull.Create ([478](#)), TJSONArray.Create ([458](#)), TJSONObject.Create ([481](#))

### 15.8.4 TJSONBoolean.JSONType

Synopsis: native JSON data type

Declaration: class function JSONType; Override

Visibility: public

Description: JSONType is overridden by TJSONString to return jtBoolean.

See also: TJSONData.JSONType ([468](#))

### 15.8.5 TJSONBoolean.Clear

Synopsis: Clear data

Declaration: procedure Clear; Override

Visibility: public

Description: Clear is overridden by TJSONBoolean to set the value to False.

See also: TJSONData.Clear ([468](#))

### 15.8.6 TJSONBoolean.Clone

Synopsis: Clone boolean value

Declaration: function Clone : TJSONData; Override

Visibility: public

Description: Clone overrides TJSONData.Clone ([471](#)) and creates an instance of the same class with the same boolean value.

See also: TJSONData.Clone ([471](#))

## 15.9 TJSONData

### 15.9.1 Description

TJSONData is an abstract class which introduces all properties and methods needed to work with JSON-based data. It should never be instantiated. Based on the type of data that must be represented one of the following descendants must be instantiated instead.

**Numbers** must be represented using one of TJSONIntegerNumber (477), TJSONFloatNumber (475) or TJSONInt64Number (476), depending on the type of the number.

**Strings** can be represented with TJSONString (489).

**Boolean** can be represented with TJSONBoolean (465).

**null** is supported using TJSONNull (478)

**Array** data can be represented using TJSONArray (457)

**Object** data can be supported using TJSONObject (480)

See also: TJSONIntegerNumber (477), TJSONString (489), TJSONBoolean (465), TJSONNull (478), TJSONArray (457), TJSONObject (480)

### 15.9.2 Method overview

Page	Property	Description
468	Clear	Clear the raw value of this data object
471	Clone	Duplicate the value of the JSON data
467	Create	Create a new instance of TJSONData.
469	FindPath	Find data by name
471	FormatJSON	Return a formatted JSON representation of the data.
468	GetEnumerator	Return an enumerator for the data
471	GetPath	Get data by name
468	JSONType	The native JSON data type represented by this object

### 15.9.3 Property overview

Page	Property	Access	Description
474	AsBoolean	rw	Access the raw JSON value as a boolean
473	AsFloat	rw	Access the raw JSON value as a float
474	AsInt64	rw	Access the raw JSON value as an 64-bit integer
473	AsInteger	rw	Access the raw JSON value as an 32-bit integer
475	AsJSON	r	Return a JSON representation of the value
472	AsString	rw	Access the raw JSON value as a string
472	Count	r	Number of sub-items for this data element
474	IsNull	r	Is the data a null value ?
472	Items	rw	Indexed access to sub-items
472	Value	rw	The value of this data object as a variant.

### 15.9.4 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 descendants should be instantiated.

See also: `TJSONIntegerNumber.Create` (478), `TJSONString.Create` (490), `TJSONBoolean.Create` (466), `TJSONNull.Create` (478), `TJSONArray.Create` (458), `TJSONObject.Create` (481)

### 15.9.5 TJSONData.JSONType

Synopsis: The native JSON data type represented by this object

Declaration: `class function JSONType; 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` (452)

### 15.9.6 TJSONData.Clear

Synopsis: Clear the raw value of this data object

Declaration: `procedure Clear; Virtual; Abstract`

Visibility: public

Description: `Clear` is implemented by the descendants 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` (447)

### 15.9.7 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` (467) type, but a `TJSONEnum` (450) structure, which contains the value, and for structured types, the key (numerical or string).

See also: `TJSONEnum` (450), `TJSONArray` (457), `TJSONObject` (480)

### 15.9.8 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 objects (480), a member can be specified using its name, and the object value itself can be retrieved with the empty path.
- For Arrays (480), the elements can be found based on an array index. The array value itself can be retrieved with the empty path.

The following code will return the value itself, i.e. E will contain the same element as D:

```
Var
  D,E : TJSONData;

begin
  D:=TJSONIntegerNumber.Create(123);
  E:=D.FindPath('');
end.
```

The following code will not return anything:

```
Var
  D,E : TJSONData;

begin
  D:=TJSONIntegerNumber.Create(123);
  E:=D.FindPath('a');
end.
```

The following code will return the third element from the array:

```
Var
  D,E : TJSONData;

begin
  D:=TJSONArray.Create([1,2,3,4,5]);
  E:=D.FindPath('[2]');
  Writeln(E.AsJSON);
end.
```

The output of this program is 3.

The following code returns the element Age from the object:

```

Var
  D,E : TJSONData;

begin
  D:=TJSONObject.Create(['Age',23,
                        'Lastame','Rodriguez',
                        'FirstName','Roberto']);
  E:=D.FindPath('Age');
  Writeln(E.AsJSON);
end.

```

The code will print 23.

Obviously, this can be combined:

```

Var
  D,E : TJSONData;

begin
  D:=TJSONObject.Create(['Age',23,
                        'Names', TJSONObject.Create([
                          'LastName','Rodriguez',
                          'FirstName','Roberto])]);
  E:=D.FindPath('Names.LastName');
  Writeln(E.AsJSON);
end.

```

And mixed:

```

var
  D,E : TJSONData;

begin
  D:=TJSONObject.Create(['Children',
    TJSONArray.Create([
      TJSONObject.Create(['Age',23,
        'Names', TJSONObject.Create([
          'LastName','Rodriguez',
          'FirstName','Roberto'])
      ]),
      TJSONObject.Create(['Age',20,
        'Names', TJSONObject.Create([
          'LastName','Rodriguez',
          'FirstName','Maria'])
      ])
    ])
  );
  E:=D.FindPath('Children[1].Names.FirstName');
  Writeln(E.AsJSON);
end.

```

See also: [TJSONArray \(457\)](#), [TJSONObject \(480\)](#), [GetPath \(447\)](#)

### **15.9.9 TJSONData.GetPath**

**Synopsis:** Get data by name

**Declaration:** `function GetPath(const APath: TJSONStringType) : TJSONData`

**Visibility:** public

**Description:** `GetPath` is identical to `FindPath` (447) but raises an exception if no element was found. The exception message contains the piece of path that was not found.

**Errors:** An EJSON (456) exception is raised if the path does not exist.

**See also:** `FindPath` (447)

### **15.9.10 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` (447)

### **15.9.11 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` (447) property, but for complex values (`TJSONArray` (457) and `TJSONObject` (480)) 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` (447), `TFormatOptions` (449)

### **15.9.12 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 ([447](#))

### **15.9.13 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 descendants (TJSONArray ([457](#)) and TJSONObject ([480](#))) override this method to return the Index-th element in the list.

See also: Count ([447](#)), TJSONArray ([457](#)), TJSONObject ([480](#))

### **15.9.14 TJSONData.Value**

**Synopsis:** The value of this data object as a variant.

**Declaration:** Property Value : variant

**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 (JSONType ([447](#))), 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: JSONType ([447](#))

### **15.9.15 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 (489) this will return the native value.

For complex values, reading or writing this property will result in an EConvertError exception.

**See also:** AsInteger (447), Value (447), AsInt64 (447), AsFloat (447), AsBoolean (447), IsNull (447), AsJSON (447)

### **15.9.16 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 (475) this will return the native value.

For complex values, reading or writing this property will always result in an EConvertError exception.

**See also:** AsInteger (447), Value (447), AsInt64 (447), AsString (447), AsBoolean (447), IsNull (447), AsJSON (447)

### **15.9.17 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 (477) this will return the native value.

For complex values, reading or writing this property will always result in an EConvertError exception.

**See also:** AsFloat (447), Value (447), AsInt64 (447), AsString (447), AsBoolean (447), IsNull (447), AsJSON (447)

### **15.9.18 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 (476) this will return the native value.

For complex values, reading or writing this property will always result in an EConvertError exception.

**See also:** AsFloat (447), Value (447), AsInteger (447), AsString (447), AsBoolean (447), IsNull (447), AsJSON (447)

### **15.9.19 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 nonzero numbers result in True, a zero results in False. If either conversion fails, an EConvertError exception is raised.

For TJSONBoolean (465) this will return the native value.

For complex values, reading or writing this property will always result in an EConvertError exception.

**See also:** AsFloat (447), Value (447), AsInt64 (447), AsString (447), AsInteger (447), IsNull (447), AsJSON (447)

### **15.9.20 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 (478) instance. In all other cases, it is False. This value cannot be set.

**See also:** TJSONType (452), JSONType (447), TJSONNull (478), AsFloat (447), Value (447), AsInt64 (447), AsString (447), AsInteger (447), AsBoolean (447), AsJSON (447)

### 15.9.21 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 array.

**See also:** AsFloat (447), Value (447), AsInt64 (447), AsString (447), AsInteger (447), AsBoolean (447), AsJSON (447)

## 15.10 TJSONFloatNumber

### 15.10.1 Description

TJSONFloatNumber must be used whenever floating point data must be represented. It can handle TJSONFloatType (447) data (normally a double). For integer data, TJSONIntegerNumber (477) or TJSONInt64Number (476) are better suited.

**See also:** TJSONIntegerNumber (477), TJSONInt64Number (476)

### 15.10.2 Method overview

Page	Property	Description
476	Clear	Clear value
476	Clone	Clone floating point value
475	Create	Create a new floating-point value
475	NumberType	Kind of numerical data managed by this class.

### 15.10.3 TJSONFloatNumber.Create

**Synopsis:** Create a new floating-point value

**Declaration:** constructor Create(AValue: TJSONFloat); Reintroduce

**Visibility:** public

**Description:** Create instantiates a new JSON floating point value, and initializes it with AValue.

**See also:** TJSONIntegerNumber.Create (478), TJSONInt64Number.Create (476)

### 15.10.4 TJSONFloatNumber.NumberType

**Synopsis:** Kind of numerical data managed by this class.

**Declaration:** class function NumberType; Override

**Visibility:** public

**Description:** NumberType is overridden by TJSONFloatNumber to return ntFloat.

**See also:** TJJSONNumberType (451), TJSONData.JSONtype (468)

### 15.10.5 TJSONFloatNumber.Clear

**Synopsis:** Clear value

**Declaration:** procedure Clear;   Override

**Visibility:** public

**Description:** Clear is overridden by TJSONFloatNumber to set the value to 0.0

See also: TJSONData.Clear ([468](#))

### 15.10.6 TJSONFloatNumber.Clone

**Synopsis:** Clone floating point value

**Declaration:** function Clone : TJSONData;   Override

**Visibility:** public

**Description:** Clone overrides TJSONData.Clone ([471](#)) and creates an instance of the same class with the same floating-point value.

See also: TJSONData.Clone ([471](#))

## 15.11 TJSONInt64Number

### 15.11.1 Description

TJSONInt64Number must be used whenever 64-bit integer data must be represented. For 32-bit integer data, TJSONIntegerNumber ([477](#)) must be used.

See also: TJSONFloatNumber ([475](#)), TJSONIntegerNumber ([477](#))

### 15.11.2 Method overview

Page	Property	Description
<a href="#">477</a>	Clear	Clear value
<a href="#">477</a>	Clone	Clone 64-bit integer value
<a href="#">476</a>	Create	Create a new instance of 64-bit integer JSON data
<a href="#">477</a>	NumberType	Kind of numerical data managed by this class.

### 15.11.3 TJSONInt64Number.Create

**Synopsis:** Create a new instance of 64-bit integer JSON data

**Declaration:** constructor Create(AValue: Int64);   Reintroduce

**Visibility:** public

**Description:** Create instantiates a new 64-bit integer JSON data and initializes the value with AValue.

See also: TJSONIntegerNumber.Create ([478](#)), TJSONFloatNumber.Create ([475](#))

### 15.11.4 TJSONInt64Number.NumberType

**Synopsis:** Kind of numerical data managed by this class.

**Declaration:** class function NumberType;   Override

**Visibility:** public

**Description:** NumberType is overridden by TJSONInt64Number to return ntInt64.

See also: TJSONNumberType (451), TJSONData.JSONtype (468)

### 15.11.5 TJSONInt64Number.Clear

**Synopsis:** Clear value

**Declaration:** procedure Clear;   Override

**Visibility:** public

**Description:** Clear is overridden by TJSONInt64Number to set the value to 0.

See also: TJSONData.Clear (468)

### 15.11.6 TJSONInt64Number.Clone

**Synopsis:** Clone 64-bit integer value

**Declaration:** function Clone : TJSONData;   Override

**Visibility:** public

**Description:** Clone overrides TJSONData.Clone (471) and creates an instance of the same class with the same 64-bit integer value.

See also: TJSONData.Clone (471)

## 15.12 TJSONIntegerNumber

### 15.12.1 Description

TJSONIntegerNumber must be used whenever 32-bit integer data must be represented. For 64-bit integer data, TJSONInt64Number (476) must be used.

See also: TJSONFloatNumber (475), TJSONInt64Number (476)

### 15.12.2 Method overview

Page	Property	Description
478	Clear	Clear value
478	Clone	Clone 32-bit integer value
478	Create	Create a new instance of 32-bit integer JSON data
478	NumberType	Kind of numerical data managed by this class.

### 15.12.3 TJSONIntegerNumber.Create

**Synopsis:** Create a new instance of 32-bit integer JSON data

**Declaration:** constructor Create(AValue: Integer); Reintroduce

**Visibility:** public

**Description:** Create instantiates a new 32-bit integer JSON data and initializes the value with AValue.

See also: TJSONFloatNumber.Create (475), TJSONInt64Number.Create (476)

### 15.12.4 TJSONIntegerNumber.NumberType

**Synopsis:** Kind of numerical data managed by this class.

**Declaration:** class function NumberType; Override

**Visibility:** public

**Description:** NumberType is overridden by TJSONIntegerNumber to return ntInteger.

See also: TJSONNumberType (451), TJSONData.JSONtype (468)

### 15.12.5 TJSONIntegerNumber.Clear

**Synopsis:** Clear value

**Declaration:** procedure Clear; Override

**Visibility:** public

**Description:** Clear is overridden by TJSONIntegerNumber to set the value to 0.

See also: TJSONData.Clear (468)

### 15.12.6 TJSONIntegerNumber.Clone

**Synopsis:** Clone 32-bit integer value

**Declaration:** function Clone : TJSONData; Override

**Visibility:** public

**Description:** Clone overrides TJSONData.Clone (471) and creates an instance of the same class with the same 32-bit integer value.

See also: TJSONData.Clone (471)

## 15.13 TJSONNull

### 15.13.1 Description

TJSONNull must be used whenever a null value must be represented.

See also: TJSONFloatNumber (475), TJSONIntegerNumber (477), TJSONInt64Number (476), TJSONBoolean (465), TJSONString (489), TJSONArray (457), TJSONObject (480)

### 15.13.2 Method overview

Page	Property	Description
<a href="#">479</a>	Clear	Clear data
<a href="#">479</a>	Clone	Clone boolean value
<a href="#">479</a>	JSONType	native JSON data type

### 15.13.3 TJSONNull.JSONType

Synopsis: native JSON data type

Declaration: class function JSONType;   Override

Visibility: public

Description: JSONType is overridden by TJSONNull to return jtNull.

See also: TJSONData.JSONType ([468](#))

### 15.13.4 TJSONNull.Clear

Synopsis: Clear data

Declaration: procedure Clear;   Override

Visibility: public

Description: Clear does nothing.

See also: TJSONData.Clear ([468](#))

### 15.13.5 TJSONNull.Clone

Synopsis: Clone boolean value

Declaration: function Clone : TJSONData;   Override

Visibility: public

Description: Clone overrides TJSONData.Clone ([471](#)) and creates an instance of the same class.

See also: TJSONData.Clone ([471](#))

## 15.14 TJSONNumber

### 15.14.1 Description

TJSONNumber is an abstract class which serves as the ancestor for the 3 numerical classes. It should never be instantiated directly. Instead, depending on the kind of data, one of TJSONIntegerNumber ([477](#)), TJSONInt64Number ([476](#)) or TJSONFloatNumber ([475](#)) should be instantiated.

See also: TJSONIntegerNumber ([477](#)), TJSONInt64Number ([476](#)), TJSONFloatNumber ([475](#))

### 15.14.2 Method overview

Page	Property	Description
480	JSONType	native JSON data type
480	NumberType	Kind of numerical data managed by this class.

### 15.14.3 TJSONNumber.JSONType

Synopsis: native JSON data type

Declaration: class function JSONType;   Override

Visibility: public

Description: JSONType is overridden by TJSONNumber to return jtNumber.

See also: TJSONData.JSONType (468)

### 15.14.4 TJSONNumber.NumberType

Synopsis: Kind of numerical data managed by this class.

Declaration: class function NumberType;   Virtual;   Abstract

Visibility: public

Description: NumberType is overridden by TJSONNumber descendants to return the kind of numerical data that can be managed by the class.

See also: TJSONIntegerNumber (477), TJSONInt64Number (476), TJSONFloatNumber (475), JSONType (447)

## 15.15 TJSONObject

### 15.15.1 Description

TJSONObjectClass is the class type of TJSONObject (480). It is used in the factory methods.

See also: TJSONObject (480), SetJSONInstanceType (455), GetJSONInstanceType (454)

### 15.15.2 Method overview

Page	Property	Description
485	Add	Add a name, value to the object
484	Clear	Clear the object
482	Clone	Clone the JSON object
481	Create	Create a new instance of JSON object data.
485	Delete	Delete an element from the list by index
482	Destroy	Free the JSON object
486	Extract	Extract an element from the object
484	Find	Find an element by name.
484	Get	Retrieve a value by name
483	GetEnumerator	Get an object enumerator
483	IndexOf	Return index of JSONData instance in object
483	IndexOfName	Return index of name in item list
483	Iterate	Iterate over all elements in the object
482	JSONType	native JSON data type
485	Remove	Remove item by instance

### 15.15.3 Property overview

Page	Property	Access	Description
489	Arrays	rw	Named access to JSON array values
488	Booleans	rw	Named access to boolean values
486	Elements	rw	Name-based access to JSON values in the object.
487	FLOATS	rw	Named access to float values
488	Int64s	rw	Named access to int64 values
487	Integers	rw	Named access to integer values
486	Names	r	Indexed access to the names of elements.
487	Nulls	rw	Named access to null values
489	Objects	rw	Named access to JSON object values
488	Strings	rw	Named access to string values
487	Types	r	Types of values in the object.

### 15.15.4 TJSONObject.Create

**Synopsis:** Create a new instance of JSON object data.

**Declaration:** constructor Create; Reintroduce  
constructor Create(const Elements: Array of const); Overload

**Visibility:** public

**Description:** Create creates a new JSON object instance, and initializes the data with Elements. Elements is an array containing an even number of items, alternating a name and a value. The names must be strings, and the values are converted to various TJSONData (467) 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 ([447](#)), GetJSONInstanceType ([454](#))

### 15.15.5 TJSONObject.Destroy

Synopsis: Free the JSON object

Declaration: destructor Destroy; Override

Visibility: public

Description: Destroy will delete all elements in the array and clean up the TJSONObject ([480](#)) instance.

See also: TJSONObject.Clear ([484](#)), TJSONObject.Create ([481](#))

### 15.15.6 TJSONObject.JSONType

Synopsis: native JSON data type

Declaration: class function JSONType; Override

Visibility: public

Description: JSONType is overridden by TJSONObject to return jtObject.

See also: TJSONData.JSONType ([468](#))

### 15.15.7 TJSONObject.Clone

Synopsis: Clone the JSON object

Declaration: function Clone : TJSONData; Override

Visibility: public

Description: Clone creates a new TJSONObject, clones all elements in the array and adds them to the newly created array with the same names as they were in the array.

See also: TJSONData.Clone ([471](#))

### 15.15.8 TJSONObject.GetEnumerator

**Synopsis:** Get an object enumerator

**Declaration:** function GetEnumerator : TBaseJSONEnumerator; Override

**Visibility:** public

**Description:** GetEnumerator is overridden in TJSONObject so it returns an object enumerator. The array enumerator will return all the elements in the array, and stores their name in the Key and index in the KeyNum members of TJSONEnum (450).

**See also:** TJSONEnum (450), TJSONData.GetEnumerator (468)

### 15.15.9 TJSONObject.Iterate

**Synopsis:** Iterate over all elements in the object

**Declaration:** procedure Iterate(Iterator: TJSONObjectIterator; Data: TObject)

**Visibility:** public

**Description:** Iterate iterates over all elements in the object, passing them one by one with name and value to the Iterator callback, together with the Data parameter. The iteration stops when all elements have been passed or when the iterator callback returned False in the Continue parameter.

**See also:** TJSONObjectIterator (451)

### 15.15.10 TJSONObject.IndexOf

**Synopsis:** Return index of JSONData instance in object

**Declaration:** function IndexOf(Item: TJSONData) : Integer

**Visibility:** public

**Description:** IndexOf compares all elements in the object with Obj and returns the index (in the TJSONData.Items (472) 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 (447), IndexOfName (447)

### 15.15.11 TJSONObject.IndexOfName

**Synopsis:** Return index of name in item list

**Declaration:** function IndexOfName(const AName: TJSONStringType;  
CaseInsensitive: Boolean) : Integer

**Visibility:** public

**Description:** IndexOfName compares the names of all elements in the object with AName and returns the index (in the TJSONData.Items (472) 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 (447)

### **15.15.12 TJSONObject.Find**

**Synopsis:** Find an element by name.

**Declaration:** function Find(const AName: string) : TJSONObject; Overload  
function Find(const AName: string; AType: TJSONType) : TJSONObject;  
; Overload

**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 ([447](#)), IndexOfName ([447](#))

### **15.15.13 TJSONObject.Get**

**Synopsis:** Retrieve a value by name

**Declaration:** function Get(const AName: string) : Variant  
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: Boolean) : Boolean  
function Get(const AName: string; ADefault: TJSONStringType)  
: TJSONStringType  
function Get(const AName: string; ADefault: TJSONArray) : TJSONArray  
function Get(const AName: string; ADefault: TJSONObject) : TJSONObject

**Visibility:** public

**Description:** Get can be used to retrieve a value by name. If an element with name equal to AName exists, and its type corresponds to the type of the ADefault, then the value is returned. If no element element with the correct type exists, the ADefault value is returned.

If no default is specified, the value is returned as a variant type, or Null if no value was found.

The other value retrieval properties such as Integers ([447](#)), Int64s ([447](#)), Booleans ([447](#)), Strings ([447](#)), Floats ([447](#)), Arrays ([447](#)), Objects ([447](#)) will raise an exception if the name is not found. The Get function does not raise an exception.

**See also:** Integers ([447](#)), Int64s ([447](#)), Booleans ([447](#)), Strings ([447](#)), Floats ([447](#)), Arrays ([447](#)), Objects ([447](#))

### **15.15.14 TJSONObject.Clear**

**Synopsis:** Clear the object

**Declaration:** procedure Clear; Override

**Visibility:** public

**Description:** Clear clears the object and frees all elements in it. After the call to Clear, Count ([472](#)) returns 0.

**See also:** Delete ([447](#)), Extract ([447](#))

### 15.15.15 TJSONObject.Add

**Synopsis:** Add a name, value to the object

```
Declaration: function Add(const AName: TJSONStringType; AValue: TJSONData) : Integer
            ; Overload
    function Add(const AName: TJSONStringType; AValue: Boolean) : Integer
            ; Overload
    function Add(const AName: TJSONStringType; AValue: TJSONFloat) : Integer
            ; Overload
    function Add(const AName: TJSONStringType; const AValue: TJSONStringType)
            : Integer; Overload
    function Add(const AName: TJSONStringType; Avalue: Integer) : Integer
            ; Overload
    function Add(const AName: TJSONStringType; Avalue: Int64) : Integer
            ; Overload
    function Add(const AName: TJSONStringType) : Integer; Overload
    function Add(const AName: TJSONStringType; AValue: TJSONArray) : Integer
            ; Overload
```

**Visibility:** public

**Description:** Add adds the value AValue with name AName to the object. If the value is not a TJSONData (467) 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 (447), Int64s (447), Booleans (447) Strings (447), Floats (447), Arrays (447) and Objects (447) 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 (447), Int64s (447), Booleans (447), Strings (447), Floats (447), Arrays (447), Objects (447)

### 15.15.16 TJSONObject.Delete

**Synopsis:** Delete an element from the list by index

```
Declaration: procedure Delete(Index: Integer)
            procedure Delete(const AName: string)
```

**Visibility:** public

**Description:** Delete deletes the element with given Index or AName from the list. The TJSONData (467) 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 (447), Add (447), Extract (447), Exchange (447)

### 15.15.17 TJSONObject.Remove

**Synopsis:** Remove item by instance

```
Declaration: procedure Remove(Item: TJSONData)
```

**Visibility:** public

**Description:** Remove will locate the value `Item` in the list of values, and removes it if it exists. The item is freed.

**See also:** Delete ([447](#)), Extract ([447](#))

### **15.15.18 TJSONObject.Extract**

**Synopsis:** Extract an element from the object

**Declaration:** function Extract(Index: Integer) : TJSONData  
function Extract(const AName: string) : TJSONData

**Visibility:** public

**Description:** Extract removes the element at position `Index` or with the `AName` from the list, just as Delete ([447](#)) 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 ([447](#)), Clear ([447](#)), Insert ([447](#)), Add ([447](#))

### **15.15.19 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 ([447](#)), Elements ([447](#))

### **15.15.20 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 ([456](#)) exception is raised.

**See also:** TJSONData.Items ([472](#)), Names ([447](#)), Types ([447](#)), Integers ([447](#)), Int64s ([447](#)), Booleans ([447](#)), Strings ([447](#)), Floats ([447](#)), Arrays ([447](#)), Objects ([447](#))

### **15.15.21 TJSONObject.Types**

**Synopsis:** Types of values in the object.

**Declaration:** Property Types [AName: string]: TJSONObjecttype

**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 ([456](#)) exception is raised.

**See also:** TJSONData.Items ([472](#)), Names ([447](#)), Elements ([447](#)), Integers ([447](#)), Int64s ([447](#)), Booleans ([447](#)), Strings ([447](#)), Floats ([447](#)), Arrays ([447](#)), Nulls ([447](#)), Objects ([447](#))

### **15.15.22 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 ([456](#)) exception is raised when reading. When writing, any existing value is replaced by a null value.

**See also:** TJSONData.Items ([472](#)), Names ([447](#)), Elements ([447](#)), Integers ([447](#)), Int64s ([447](#)), Booleans ([447](#)), Strings ([447](#)), Floats ([447](#)), Arrays ([447](#)), Types ([447](#)), Objects ([447](#))

### **15.15.23 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 ([456](#)) exception is raised when reading. When writing, any existing value is replaced by the specified floating-point value.

**See also:** TJSONData.Items ([472](#)), Names ([447](#)), Elements ([447](#)), Integers ([447](#)), Int64s ([447](#)), Booleans ([447](#)), Strings ([447](#)), Nulls ([447](#)), Arrays ([447](#)), Types ([447](#)), Objects ([447](#))

### **15.15.24 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 (456) exception is raised when reading. When writing, any existing value is replaced by the specified integer value.

**See also:** TJSONData.Items (472), Names (447), Elements (447), Floats (447), Int64s (447), Booleans (447), Strings (447), Nulls (447), Arrays (447), Types (447), Objects (447)

### 15.15.25 TJSONObject.Int64s

**Synopsis:** Named access to int64 values

**Declaration:** Property Int64s [AName: string]: Int64

**Visibility:** public

**Access:** Read,Write

**Description:** Int64s allows to retrieve or set the int64 values in the object by name. If a non-existent name is specified, an EJSON (456) exception is raised when reading. When writing, any existing value is replaced by the specified int64 value.

**See also:** TJSONData.Items (472), Names (447), Elements (447), Floats (447), Integers (447), Booleans (447), Strings (447), Nulls (447), Arrays (447), Types (447), Objects (447)

### 15.15.26 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 (456) exception is raised when reading. When writing, any existing value is replaced by the specified string value.

**See also:** TJSONData.Items (472), Names (447), Elements (447), Floats (447), Integers (447), Booleans (447), Int64s (447), Nulls (447), Arrays (447), Types (447), Objects (447)

### 15.15.27 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 (456) exception is raised when reading. When writing, any existing value is replaced by the specified boolean value.

**See also:** TJSONData.Items (472), Names (447), Elements (447), Floats (447), Integers (447), Strings (447), Int64s (447), Nulls (447), Arrays (447), Types (447), Objects (447)

### 15.15.28 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 (456) exception is raised when reading. When writing, any existing value is replaced by the specified JSON array.

**See also:** [TJSONData.Items](#) (472), [Names](#) (447), [Elements](#) (447), [Floats](#) (447), [Integers](#) (447), [Strings](#) (447), [Int64s](#) (447), [Nulls](#) (447), [Booleans](#) (447), [Types](#) (447), [Objects](#) (447)

### 15.15.29 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 (456) exception is raised when reading. When writing, any existing value is replaced by the specified JSON object.

**See also:** [TJSONData.Items](#) (472), [Names](#) (447), [Elements](#) (447), [Floats](#) (447), [Integers](#) (447), [Strings](#) (447), [Int64s](#) (447), [Nulls](#) (447), [Booleans](#) (447), [Types](#) (447), [Arrays](#) (447)

## 15.16 TJSONString

### 15.16.1 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](#) (475), [TJSONIntegerNumber](#) (477), [TJSONInt64Number](#) (476), [TJSONBoolean](#) (465), [TJSONNull](#) (478), [TJSONArray](#) (457), [TJSONObject](#) (480)

### 15.16.2 Method overview

Page	Property	Description
<a href="#">490</a>	Clear	Clear value
<a href="#">490</a>	Clone	Clone string value
<a href="#">490</a>	Create	Create a new instance of string JSON data
<a href="#">490</a>	JSONType	native JSON data type

### **15.16.3 TJSONString.Create**

**Synopsis:** Create a new instance of string JSON data

**Declaration:** constructor Create(const AValue: TJSONStringType); Reintroduce

**Visibility:** public

**Description:** Create instantiates a new string JSON data and initializes the value with AValue. Currently the implementation uses an ansi string to hold the data. This means that to correctly hold unicode data, a UTF-8 encoding must be used.

**See also:** TJSONIntegerNumber.Create ([478](#)), TJSONFloatNumber.Create ([475](#)), TJSONInt64Number.Create ([476](#)), TJSONBoolean.Create ([466](#)), TJSONNull.Create ([478](#)), TJSONArray.Create ([458](#)), TJSONObject.Create ([481](#))

### **15.16.4 TJSONString.JSONType**

**Synopsis:** native JSON data type

**Declaration:** class function JSONType; Override

**Visibility:** public

**Description:** JSONType is overridden by TJSONString to return jtString.

**See also:** TJSONData.JSONType ([468](#))

### **15.16.5 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 ([468](#))

### **15.16.6 TJSONString.Clone**

**Synopsis:** Clone string value

**Declaration:** function Clone : TJSONData; Override

**Visibility:** public

**Description:** Clone overrides TJSONData.Clone ([471](#)) and creates an instance of the same class with the same string value.

**See also:** TJSONData.Clone ([471](#))

# Chapter 16

## Reference for unit 'fpTimer'

### 16.1 Used units

Table 16.1: Used units by unit 'fpTimer'

Name	Page
Classes	??
System	??

### 16.2 Overview

The fpTimer unit implements a timer class TFPTimer (493) which can be used on all supported platforms. The timer class uses a driver class TFPTimerDriver (494) 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 (491) variable to the class pointer of the driver class. The driver class should descend from TFPTimerDriver (494).

### 16.3 Constants, types and variables

#### 16.3.1 Types

```
TFPTimerDriverClass = Class of TFPTimerDriver
```

TFPTimerDriverClass is the class pointer of TFPTimerDriver (494) it exists mainly for the purpose of being able to set DefaultTimerDriverClass (491), so a custom timer driver can be used for the timer instances.

#### 16.3.2 Variables

```
DefaultTimerDriverClass : TFPTimerDriverClass = Nil
```

`DefaultTimerDriverClass` contains the `TFPTimerDriver` (494) class pointer that should be used when a new instance of `TFPCustomTimer` (492) 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.

## 16.4 TFPCustomTimer

### 16.4.1 Description

`TFPCustomTimer` is the timer class containing the timer's implementation. It relies on an extra driver instance (of type `TFPTimerDriver` (494)) to do the actual work.

`TFPCustomTimer` publishes no events or properties, so it is unsuitable for handling in an IDE. The `TFPTimer` (493) descendent class publishes all needed events of `TFPCustomTimer`.

See also: `TFPTimerDriver` (494), `TFPTimer` (493)

### 16.4.2 Method overview

Page	Property	Description
492	Create	Create a new timer
492	Destroy	Release a timer instance from memory
493	StartTimer	Start the timer
493	StopTimer	Stop the timer

### 16.4.3 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` (492)

### 16.4.4 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` (492)

### 16.4.5 **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 ([493](#)) event is called.

**See also:** StopTimer ([493](#)), Enabled ([493](#)), OnTimer ([494](#))

### 16.4.6 **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 ([493](#)), Enabled ([493](#)), OnTimer ([494](#))

## 16.5 **TFPTimer**

### 16.5.1 **Description**

TFPTimer implements no new events or properties, but merely publishes events and properties already implemented in TFPCustomTimer ([492](#)): Enabled ([493](#)), OnTimer ([494](#)) and Interval ([494](#)).

The TFPTimer class is suitable for use in an IDE.

**See also:** TFPCustomTimer ([492](#)), Enabled ([493](#)), OnTimer ([494](#)), Interval ([494](#))

### 16.5.2 **Property overview**

Page	Property	Access	Description
<a href="#">493</a>	Enabled		Start or stop the timer
<a href="#">494</a>	Interval		Timer tick interval in milliseconds.
<a href="#">494</a>	OnTimer		Event called on each timer tick.

### 16.5.3 **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 ([493](#))), setting it to False will stop the timer (calling StopTimer ([493](#))).

**See also:** StartTimer ([493](#)), StopTimer ([493](#)), OnTimer ([494](#)), Interval ([494](#))

### **16.5.4 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 On-Timer ([494](#)) 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 ([494](#)), Enabled ([493](#))

### **16.5.5 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 ([494](#)), Enabled ([493](#))

## **16.6 TFPTimerDriver**

### **16.6.1 Description**

TFPTimerDriver is the abstract timer driver class: it simply provides an interface for the TFP-CustomTimer ([492](#)) class to use.

The fpTimer unit implements a descendent of this class which implements the default timer mechanism.

**See also:** TFPCustomTimer ([492](#)), DefaultTimerDriverClass ([491](#))

### **16.6.2 Method overview**

Page	Property	Description
<a href="#">495</a>	Create	Create new driver instance
<a href="#">495</a>	StartTimer	Start the timer
<a href="#">495</a>	StopTimer	Stop the timer

### **16.6.3 Property overview**

Page	Property	Access	Description
<a href="#">495</a>	Timer	r	Timer tick

#### 16.6.4 TFPTimerDriver.Create

Synopsis: Create new driver instance

Declaration: constructor Create(ATimer: TFPCustomTimer); Virtual

Visibility: public

Description: Create should be overridden by descendants of TFPTimerDriver to do additional initialization of the timer driver. Create just stores (in Timer (495)) a reference to the ATimer instance which created the driver instance.

See also: Timer (495), TFPTimer (493)

#### 16.6.5 TFPTimerDriver.StartTimer

Synopsis: Start the timer

Declaration: procedure StartTimer; Virtual; Abstract

Visibility: public

Description: StartTimer is called by TFPCustomTimer.StartTimer (493). It should be overridden by descendants of TFPTimerDriver to actually start the timer.

See also: TFPCustomTimer.StartTimer (493), TFPTimerDriver.StopTimer (495)

#### 16.6.6 TFPTimerDriver.StopTimer

Synopsis: Stop the timer

Declaration: procedure StopTimer; Virtual; Abstract

Visibility: public

Description: StopTimer is called by TFPCustomTimer.StopTimer (493). It should be overridden by descendants of TFPTimerDriver to actually stop the timer.

See also: TFPCustomTimer.StopTimer (493), TFPTimerDriver.StartTimer (495)

#### 16.6.7 TFPTimerDriver.Timer

Synopsis: Timer tick

Declaration: Property Timer : TFPCustomTimer

Visibility: public

Access: Read

Description: Timer calls the TFPCustomTimer (492) timer event. Descendents of TFPTimerDriver should call Timer whenever a timer tick occurs.

See also: TFPTimer.OnTimer (494), TFPTimerDriver.StartTimer (495), TFPTimerDriver.StopTimer (495)

# Chapter 17

## Reference for unit 'gettext'

### 17.1 Used units

Table 17.1: Used units by unit 'gettext'

Name	Page
Classes	??
System	??
sysutils	??

### 17.2 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` (498)) 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.

### 17.3 Constants, types and variables

#### 17.3.1 Constants

`MOFileHeaderMagic` = \$950412de

This constant is found as the first integer in a `.mo`

#### 17.3.2 Types

`PLongWordArray` = `^TLongWordArray`

Pointer to a `TLongWordArray` (497) array.

`PMOStringTable` = `^TMOStringTable`

Pointer to a TMOSTringTable (497) array.

```
PPCharArray = ^TPCharArray
```

Pointer to a TPCharArray (497) array.

```
TLongWordArray = Array[0..(1sh130) divSizeOf(LongWord)] of LongWord
```

TLongWordArray is an array used to define the PLongWordArray (496) 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.

```
TMOFfileHeader = 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.

```
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 TMOFfileHeader (497) record at the start of the file.

```
TMOSTringTable = Array[0..(1sh130) divSizeOf(TMOSTringInfo)] of TMOSTringInfo
```

TMOSTringTable is an array type containing TMOSTringInfo (497) records. It should never be used directly, as it would occupy too much memory.

```
TPCharArray = Array[0..(1sh130) divSizeOf(PChar)] of PChar
```

TLongWordArray is an array used to define the PPCharArray (497) 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.

## 17.4 Procedures and functions

### 17.4.1 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.

### 17.4.2 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 (498) 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 (498), TMOFile (498)

### 17.4.3 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 (498), 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 (498), TMOFile (498)

## 17.5 EMOFileError

### 17.5.1 Description

EMOFileError is raised in case an TMOFile (498) instance is created with an invalid .mo.

**See also:** TMOFile (498)

## 17.6 TMOFile

### 17.6.1 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.

## 17.6.2 Method overview

Page	Property	Description
499	Create	Create a new instance of the TMOFile class.
499	Destroy	Removes the TMOFile instance from memory
499	Translate	Translate a string

## 17.6.3 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 (497) structure, then an EMOFileError (498) exception is raised.

See also: TMOFile.Destroy (499)

## 17.6.4 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 (499)

## 17.6.5 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 18

## Reference for unit 'IBConnection'

### 18.1 Used units

Table 18.1: Used units by unit 'IBConnection'

Name	Page
bufdataset	??
Classes	??
db	<a href="#">231</a>
dbconst	??
ibase60dyn	??
sqldb	<a href="#">630</a>
System	??
sysutils	??

### 18.2 Constants, types and variables

#### 18.2.1 Constants

DEFDIACLECT = 3

Default dialect that will be used when connecting to databases. See [TIBConnection.Dialect \(504\)](#) for more details on dialects.

MAXBLOBSEGMENTSIZE = 65535

#### 18.2.2 Types

```
TDatabaseInfo = record
  Dialect : Integer;
  ODSMajorVersion : Integer;
  ServerVersion : string;
  ServerVersionString : string;
end
```

## 18.3 EIBDatabaseError

### 18.3.1 Description

Firebird/Interbase database error, a descendant of db.EDatabaseError ([500](#)).

See also: db.EDatabaseError ([500](#))

## 18.4 TIBConnection

### 18.4.1 Description

TIBConnection is a descendant of TSQlConnection ([500](#)) 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, libfb-client.so, fbclient.dll, fbembed.dll) and its dependencies to be installed on the system. The bitness between library and your application must match: e.g. use 32 bit fbclient when developing a 32 bit application on 64 bit Linux.

On Windows, in accordance with the regular Windows way of loading DLLs, the library can also be in the executable directory. In fact, this directory is searched first, and might be a good option for distributing software to end users as it eliminates problems with incompatible DLL versions.

TIBConnection is based on FPC Interbase/Firebird code (ibase60.inc) that tries to load the client library. If you want to use Firebird embedded, make sure the embedded library is searched/loaded first. There are several ways to do this:

- Include `ibase60` in your uses clause, set `UseEmbeddedFirebird` to true
- On Windows, with FPC newer than 2.5.1, put `fbembed.dll` in your application directory
- On Windows, put the `fbembed.dll` in your application directory and rename it to `fbclient.dll`

Pre 2.5.1 versions of FPC did not try to load the `fbembed` library by default. See [FPC bug 17664](#) for more details.

An indication of which DLLs need to be installed on Windows (Firebird 2.5, differs between versions):

- `fbclient.dll` (or `fbembed.dll`)
- `firebird.msg`
- `ib_util.dll`
- `icudt30.dll`
- `icuin30.dll`
- `icuuc30.dll`
- `msvcp80.dll`
- `msvcr80.dll`

Please see your database documentation for details.

The `TIBConnection` component does not reliably detect computed fields as such. This means that automatically generated update sql statements will attempt to update these fields, resulting in SQL errors. These errors can be avoided by removing the `pfInUpdate` flag from the provideroptions from a field, once it has been created:

```
MyQuery.FieldByName('full_name').ProviderFlags:=[ ];
```

See also: [TSQLConnection \(500\)](#)

#### 18.4.2 Method overview

Page	Property	Description
<a href="#">502</a>	Create	Creates a <code>TIBConnection</code> object
<a href="#">502</a>	CreateDB	Creates a database on disk
<a href="#">503</a>	DropDB	Deletes a database from disk
<a href="#">502</a>	GetConnectionString	

#### 18.4.3 Property overview

Page	Property	Access	Description
<a href="#">503</a>	BlobSegmentSize	rw	Write this amount of bytes per BLOB segment
<a href="#">504</a>	DatabaseName		Name of the database to connect to
<a href="#">504</a>	Dialect	rws	Database dialect
<a href="#">504</a>	KeepConnection		Keep open connection after first query
<a href="#">505</a>	LoginPrompt		Switch for showing custom login prompt
<a href="#">504</a>	ODSMajorVersion	r	
<a href="#">505</a>	OnLogin		Event triggered when a login prompt needs to be shown.
<a href="#">505</a>	Params		Firebird/Interbase specific parameters

#### 18.4.4 TIBConnection.Create

Synopsis: Creates a `TIBConnection` object

Declaration: constructor Create(AOwner: TComponent); Override

Visibility: public

Description: Creates a `TIBConnection` object

#### 18.4.5 TIBConnection.GetConnectionString

Declaration: function GetConnectionString(InfoType: TConnInfoType) : string; Override

Visibility: public

#### 18.4.6 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 ([500](#)) (specifically, PAGE\_SIZE) and TSQLConnection.CharSet ([500](#)) 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 ([500](#)), TSQLConnection.UserName ([500](#)), TSQLConnection.Password ([500](#)).

If creating a database using Firebird embedded, make sure the embedded library is loaded, the TSQLConnection.HostName ([500](#)) property is empty, and set the TSQLConnection.UserName ([500](#)) to e.g. 'SYSDBA'. See [TIBConnection: Firebird/Interbase specific TSQLConnection](#) ([500](#)) for details on loading the embedded database library.

See also: TSQLConnection.Params ([500](#)), TSQLConnection.DropDB ([500](#)), TIBConnection ([501](#))

#### 18.4.7 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 ([501](#)).

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 ([500](#)), TSQLConnection.UserName ([500](#)), TSQLConnection.Password ([500](#)).

When using Firebird embedded, make sure the embedded connection library is loaded, the TSQLConnection.HostName ([500](#)) property is empty, and set the TSQLConnection.UserName ([500](#)) to e.g. 'SYSDBA'. See [TIBConnection: Firebird/Interbase specific TSQLConnection](#) ([500](#)) for more details on loading the embedded library.

See also: TSQLConnection.CreateDB ([500](#)), TSQLConnectionHostName ([500](#)), TSQLConnection.UserName ([500](#)), TSQLConnection.Password ([500](#))

#### 18.4.8 TIBConnection.BlobSegmentSize

Synopsis: Write this amount of bytes per BLOB segment

Declaration: Property BlobSegmentSize : Word; deprecate;

Visibility: public

Access: Read,Write

Description: **Deprecated** since FPC 2.7.1 revision 19659

When sending BLOBS to the database, the code writes them in segments.

Before FPC 2.7.1 revision 19659, these segments were 80 bytes and could be changed using BlobSegmentSize. Please set BlobSegmentSize to 65535 for better write performance.

In newer FPC versions, the BlobSegmentSize property is ignored and segments of 65535 bytes are always used.

#### **18.4.9 TIBConnection.ODSMajorVersion**

**Declaration:** Property ODSMajorVersion : Integer

**Visibility:** public

**Access:** Read

#### **18.4.10 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 filesystem, 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.

#### **18.4.11 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.

#### **18.4.12 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.

### 18.4.13 TIBConnection.LoginPrompt

**Synopsis:** Switch for showing custom login prompt

**Declaration:** Property LoginPrompt :

Visibility: published

Access:

Description: If true, the OnLogin ([500](#)) event will fire, allowing you to handle supplying of credentials yourself.

See also: TSQLEConnection.OnLogin ([500](#))

### 18.4.14 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.TSQLEConnection.Params ([660](#))

### 18.4.15 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 ([500](#)) property is True, after the BeforeConnect ([275](#)) event, but before the connection is actually established.

See also: #fcl.db.TCustomConnection.BeforeConnect ([275](#)), TSQLEConnection.LoginPrompt ([500](#)), #fcl.db.TCustomConnection.Open ([272](#)), TSQLEConnection.OnLogin ([500](#))

## 18.5 TIBConnectionDef

### 18.5.1 Description

Child of TConnectionDef (500) 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 (500)

### 18.5.2 Method overview

Page	Property	Description
506	ConnectionClass	Firebird/Interbase child of ConnectionClass (642)
507	DefaultLibraryName	
506	Description	Description for the Firebird/Interbase child of #fcl.sqldb.TConnectionDef.ConnectionClass (642)
507	LoadedLibraryName	
507	LoadFunction	
506	TypeName	Firebird/Interbase child of TConnectionDef.TypeName (500)
507	UnLoadFunction	

### 18.5.3 TIBConnectionDef.TypeName

Synopsis: Firebird/Interbase child of TConnectionDef.TypeName (500)

Declaration: class function TypeName;   Override

Visibility: default

See also: TConnectionDef.TypeName (500), TIBConnection (501)

### 18.5.4 TIBConnectionDef.ConnectionClass

Synopsis: Firebird/Interbase child of ConnectionClass (642)

Declaration: class function ConnectionClass;   Override

Visibility: default

See also: TConnectionDef.ConnectionClass (500), TIBConnection (501)

### 18.5.5 TIBConnectionDef.Description

Synopsis: Description for the Firebird/Interbase child of #fcl.sqldb.TConnectionDef.ConnectionClass (642)

Declaration: class function Description;   Override

Visibility: default

Description: The description identifies this ConnectionDef object as a Firebird/Interbase connection.

See also: #fcl.sqldb.TConnectionDef.Description (642), TIBConnection (501)

### **18.5.6 TIBConnectionDef.DefaultLibraryName**

Declaration: class function DefaultLibraryName;   Override

Visibility: default

### **18.5.7 TIBConnectionDef.LoadFunction**

Declaration: class function LoadFunction;   Override

Visibility: default

### **18.5.8 TIBConnectionDef.UnLoadFunction**

Declaration: class function UnLoadFunction;   Override

Visibility: default

### **18.5.9 TIBConnectionDef.LoadedLibraryName**

Declaration: class function LoadedLibraryName;   Override

Visibility: default

## **18.6 TIBCursor**

### **18.6.1 Description**

A cursor that keeps track of where you are in a Firebird/Interbase dataset. It is a descendent of TSQLCursor ([500](#)).

See also: TSQLCursor ([500](#)), TIBConnection ([501](#))

## **18.7 TIBTrans**

### **18.7.1 Description**

Firebird/Interbase database transaction object. Descendant of TSQLHandle ([500](#)).

See also: TSQLHandle ([500](#)), TIBConnection ([501](#))

# Chapter 19

## Reference for unit 'idea'

### 19.1 Used units

Table 19.1: Used units by unit 'idea'

Name	Page
Classes	??
System	??
sysutils	??

### 19.2 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 decription stream which decrypts anything that is read from it.

### 19.3 Constants, types and variables

#### 19.3.1 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

### 19.3.2 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` (510) or `DeKeyIDEA` (509) 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.

## 19.4 Procedures and functions

### 19.4.1 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` (510), `DeKeyIdea` (509), `TIDEAEEncryptStream` (512), `TIDEADecryptStream` (510)

### 19.4.2 DeKeyIdea

**Synopsis:** Create a decryption key from an encryption key.

**Declaration:** `procedure DeKeyIdea(z: TIDEAKey; out dk: TIDEAKey)`

**Visibility:** default

**Description:** `DeKeyIdea` creates a decryption key based on the encryption key `z`. The decryption key is returned in `dk`. Note that only a decryption key generated from the encryption key that was used to encrypt the data can be used to decrypt the data.

**Errors:** None.

See also: `EnKeyIdea` (510), `CipherIdea` (509)

### 19.4.3 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` (509), `CipherIdea` (509)

## 19.5 EIDEAError

### 19.5.1 Description

`EIDEAError` is used to signal errors in the IDEA encryption decryption streams.

## 19.6 TIDEADeCryptStream

### 19.6.1 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` (509) 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` (512) encryption stream.

See also: `TIDEAEncryptStream` (512), `TIDEAStream.Create` (514), `CipherIDEA` (509)

### 19.6.2 Method overview

Page	Property	Description
511	Create	Constructor to create a new <code>TIDEADeCryptStream</code> instance
511	Read	Reads data from the stream, decrypting it as needed
511	Seek	Set position on the stream

### 19.6.3 TIDEADeCryptStream.Create

**Synopsis:** Constructor to create a new TIDEADeCryptStream instance

**Declaration:** constructor Create(const AKey: string; Dest: TStream); Overload

**Visibility:** public

**Description:** Create creates a new TIDEADeCryptStream instance using the the string AKey to compute the encryption key (509), which is then passed on to the inherited constructor TIDEAStream.Create (514). It is an easy-access function which introduces no new functionality.

The string is truncated at the maximum length of the TIdeaCryptKey (509) structure, so it makes no sense to provide a string with length longer than this structure.

**See also:** TIdeaCryptKey (509), TIDEAStream.Create (514), TIDEAEncryptStream.Create (512)

### 19.6.4 TIDEADeCryptStream.Read

**Synopsis:** Reads data from the stream, decrypting it as needed

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

**Visibility:** public

**Description:** Read attempts to read Count bytes from the stream, placing them in Buffer the bytes are read from the source stream and decrypted as they are read. (bytes are read from the source stream in blocks of 8 bytes. The function returns the number of bytes actually read.

**Errors:** If an error occurs when reading data from the source stream, an exception may be raised.

**See also:** Write (510), Seek (511), TIDEAEncryptStream (512)

### 19.6.5 TIDEADeCryptStream.Seek

**Synopsis:** Set position on the stream

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

**Visibility:** public

**Description:** Seek will only work on a forward seek. It emulates a forward seek by reading and discarding bytes from the input stream. The TIDEADeCryptStream stream tries to provide seek capabilities for the following limited number of cases:

**Origin=soFromBeginning** If Offset is larger than the current position, then the remaining bytes are skipped by reading them from the stream and discarding them.

**Origin=soFromCurrent** If Offset is zero, the current position is returned. If it is positive, then Offset bytes are skipped by reading them from the stream and discarding them.

**Errors:** An EIDEAError (510) exception is raised if the stream does not allow the requested seek operation.

**See also:** Read (511)

## 19.7 TIDEAEncryptStream

### 19.7.1 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 (509) 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 (510) decryption stream.

See also: TIDEADecryptStream (510), TIDEAStream.Create (514), CipherIDEA (509)

### 19.7.2 Method overview

Page	Property	Description
512	Create	Constructor to create a new TIDEAEncryptStream instance
512	Destroy	Flush data buffers and free the stream instance.
513	Flush	Write remaining bytes from the stream
513	Seek	Set stream position
513	Write	Write bytes to the stream to be encrypted

### 19.7.3 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 (509), which is then passed on to the inherited constructor TIDEAStream.Create (514). It is an easy-access function which introduces no new functionality.

The string is truncated at the maximum length of the TIdeaCryptKey (509) structure, so it makes no sense to provide a string with length longer than this structure.

See also: TIdeaCryptKey (509), TIDEAStream.Create (514), TIDEADeCryptStream.Create (511)

### 19.7.4 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 (514)

### 19.7.5 TIDEAEncryptStream.Write

**Synopsis:** Write bytes to the stream to be encrypted

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

**Visibility:** public

**Description:** Write writes Count bytes from Buffer to the stream, encrypting the bytes as they are written (encryption in blocks of 8 bytes).

**Errors:** If an error occurs writing to the destination stream, an error may occur.

**See also:** Read ([511](#))

### 19.7.6 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 ([510](#)) will be raised unless called with 0 and soFromCurrent as arguments.

**See also:** Write ([513](#)), EIDEAError ([510](#))

### 19.7.7 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 ([513](#))

## 19.8 TIDEAStream

### 19.8.1 Description

Do not create instances of TIDEAStream directly. It implements no useful functionality: it serves as a common ancestor of the TIDEAEncryptStream ([512](#)) and TIDEADECryptStream ([510](#)), 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 ([512](#)), TIDEADECryptStream ([510](#))

### 19.8.2 Method overview

Page	Property	Description
<a href="#">514</a>	Create	Creates a new instance of the TIDEAStream class

### 19.8.3 Property overview

Page	Property	Access	Description
<a href="#">514</a>	Key	r	Key used when encrypting/decrypting

### 19.8.4 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 ([512](#)), TIDEADeCryptStream ([510](#))

### 19.8.5 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 ([509](#))

# Chapter 20

## Reference for unit 'inicol'

### 20.1 Used units

Table 20.1: Used units by unit 'inicol'

Name	Page
Classes	??
IniFiles	<a href="#">525</a>
System	??
sysutils	??

### 20.2 Overview

inicol contains an implementation of `TCollection` and `TCollectionItem` descendants which cooperate to read and write the collection from and to a .ini file. It uses the `TCustomIniFile` ([525](#)) class for this.

### 20.3 Constants, types and variables

#### 20.3.1 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.

## 20.4 EIniCol

### 20.4.1 Description

EIniCol is used to report error conditions in the load and save methods of TIniCollection (516).

## 20.5 TIniCollection

### 20.5.1 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 descendants of the type TIniCollectionItem (520).

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 (518), TIniCollection.LoadFromIni (518), TIniCollection.SaveToIni (517), TIniCollection.SaveToFile (517)

### 20.5.2 Method overview

Page	Property	Description
516	Load	Loads the collection from the default filename.
518	LoadFromFile	Load collection from file.
518	LoadFromIni	Load collection from a file in .ini file format.
517	Save	Save the collection to the default filename.
517	SaveToFile	Save collection to a file in .ini file format
517	SaveToIni	Save the collection to a TCustomInifile descendent

### 20.5.3 Property overview

Page	Property	Access	Description
519	FileName	rw	Filename of the collection
519	GlobalSection	rw	Name of the global section
518	Prefix	r	Prefix used in global section
519	SectionPrefix	r	Prefix string for section names

### 20.5.4 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 (519) property. It calls the LoadFromFile (518) method to do this.

**Errors:** If the collection was not loaded or saved to file before this call, an EIniCol exception will be raised.

See also: [TIniCollection.LoadFromFile](#) (518), [TIniCollection.LoadFromIni](#) (518), [TIniCollection.Save](#) (517), [FileName](#) (519)

### 20.5.5 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](#) (519) property, using [GlobalSection](#) (519) as the section. It calls the [SaveToFile](#) (517) 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](#) (517), [TIniCollection.SaveToIni](#) (517), [TIniCollection.Load](#) (516), [FileName](#) (519)

### 20.5.6 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](#) (518) property for the key. After this it calls the [SaveToIni](#) (520) method of all [TIniCollectionItem](#) (520) 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](#) (519) property contains the value of [Section](#), it will be used in the [Save](#) (520) method.

See also: [TIniCollectionItem.SaveToIni](#) (520)

### 20.5.7 TIniCollection.SaveToFile

**Synopsis:** Save collection to a file in .ini file format

**Declaration:** procedure SaveToFile(AFileName: string; Section: string)

**Visibility:** public

**Description:** `SaveToFile` will create a `TMemIniFile` instance with the `AFileName` argument as a filename. This instance is passed on to the `SaveToIni` (517) method, together with the `Section` argument, to do the actual saving.

**Errors:** An exception may be raised if the path in `AFileName` does not exist.

**See also:** `TIniCollection.SaveToIni` (517), `TIniCollection.LoadFromFile` (518)

### 20.5.8 TIniCollection.LoadFromIni

**Synopsis:** Load collection from a file in .ini file format.

**Declaration:** `procedure LoadFromIni(Ini: TCustomIniFile; Section: string); Virtual`

**Visibility:** public

**Description:** `LoadFromIni` will load the collection from the `Ini` instance. It first clears the collection, and reads the number of items from the global section with the name as passed through the `Section` argument. After this, an item is created and added to the collection, and its data is read by calling the `TIniCollectionItem.LoadFromIni` (520) 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` (517) method description.

**See also:** `TIniCollection.LoadFromFile` (518), `TIniCollectionItem.LoadFromIni` (520), `TIniCollection.SaveToIni` (517)

### 20.5.9 TIniCollection.LoadFromFile

**Synopsis:** Load collection from file.

**Declaration:** `procedure LoadFromFile(AFileName: string; Section: string)`

**Visibility:** public

**Description:** `LoadFromFile` creates a `TMemIniFile` instance using `AFileName` as the filename. It calls `LoadFromIni` (518) using this instance and `Section` as the parameters.

**See also:** `TIniCollection.LoadFromIni` (518), `TIniCollection.Load` (516), `TIniCollection.SaveToIni` (517), `TIniCollection.SaveToFile` (517)

### 20.5.10 TIniCollection.Prefix

**Synopsis:** Prefix used in global section

**Declaration:** `Property Prefix : string`

**Visibility:** public

**Access:** Read

**Description:** `Prefix` is used when writing the section names of the items in the collection to the global section, or when reading the names from the global section. If the prefix is set to `Item` then the global section might look something like this:

```
[MyCollection]
Count=2
Item1=FirstItem
Item2=SecondItem
```

A descendent of TIniCollection should set the value of this property, it cannot be empty.

See also: [TIniCollection.SectionPrefix \(519\)](#), [TIniCollection.GlobalSection \(519\)](#)

### **20.5.11 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 as returned by the TIniCollectionItem.SectionName ([521](#)) property to return the exact section name. It can be empty.

See also: [TIniCollection.Section \(516\)](#), [TIniCollection.GlobalSection \(519\)](#)

### **20.5.12 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 ([518](#)) or SaveToFile ([517](#)) operation. It is used in the Load ([516](#)) or Save ([517](#)) calls.

See also: [Save \(517\)](#), [LoadFromFile \(518\)](#), [SaveToFile \(517\)](#), [Load \(516\)](#)

### **20.5.13 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 ([518](#)) or SaveToIni ([517](#)) calls. It's used in the Load ([516](#)) or Save ([517](#)) calls.

See also: [Save \(517\)](#), [LoadFromFile \(518\)](#), [SaveToFile \(517\)](#), [Load \(516\)](#)

## 20.6 TIniCollectionItem

### 20.6.1 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` (520) and `LoadFromIni` (520) methods should be overridden. They should implement the actual loading and saving. The loading and saving is always initiated by the methods in `TIniCollection` (516), `TIniCollection.LoadFromIni` (518) and `TIniCollection.SaveToIni` (517) respectively.

See also: `TIniCollection` (516), `TIniCollectionItem.SaveToIni` (520), `TIniCollectionItem.LoadFromIni` (520), `TIniCollection.LoadFromIni` (518), `TIniCollection.SaveToIni` (517)

### 20.6.2 Method overview

Page	Property	Description
521	<code>LoadFromFile</code>	Load item from a file
520	<code>LoadFromIni</code>	Method called when the item must be loaded
521	<code>SaveToFile</code>	Save item to a file
520	<code>SaveToIni</code>	Method called when the item must be saved

### 20.6.3 Property overview

Page	Property	Access	Description
521	<code>SectionName</code>	rw	Default section name

### 20.6.4 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` (517) 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` (517), `LoadFromIni` (520), `SaveToFile` (521), `LoadFromFile` (521)

### 20.6.5 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 (518) 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 (518), SaveToIni (520), LoadFromFile (521), SaveToFile (521)

### 20.6.6 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 (520) 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 (520), LoadFromFile (521)

### 20.6.7 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 (520) to load the item from the indicated file in .ini format from the section Section.

**Errors:** None.

**See also:** SaveToFile (521), LoadFromIni (520)

### 20.6.8 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 descendants to determine a unique section name within the .ini file.

**See also:** SaveToFile (521), LoadFromIni (520)

## 20.7 TNamedIniCollection

### 20.7.1 Description

TNamedIniCollection is the collection to go with the TNamedIniCollectionItem ([523](#)) item class. It provides some functions to look for items based on the UserData ([522](#)) or based on the Name ([522](#)).

See also: TNamedIniCollectionItem ([523](#)), IndexOfUserData ([522](#)), IndexOfName ([522](#))

### 20.7.2 Method overview

Page	Property	Description
<a href="#">523</a>	FindByName	Return the item based on its name
<a href="#">523</a>	FindByUserData	Return the item based on its UserData
<a href="#">522</a>	IndexOfName	Search for an item, based on its name, and return its position
<a href="#">522</a>	IndexOfUserData	Search for an item based on its UserData property

### 20.7.3 Property overview

Page	Property	Access	Description
<a href="#">523</a>	NamedItems	rw	Indexed access to the TNamedIniCollectionItem items

### 20.7.4 TNamedIniCollection.IndexOfUserData

Synopsis: Search for an item based on its UserData property

Declaration: `function IndexOfUserData(UserData: TObject) : Integer`

Visibility: public

Description: `IndexOfUserData` searches the list of items and returns the index of the item which has `UserData` in its `UserData` ([522](#)) property. If no such item exists, -1 is returned.

Note that the (linear) search starts at the last element and works it's way back to the first.

Errors: If no item exists, -1 is returned.

See also: IndexOfName ([522](#)), TNamedIniCollectionItem.UserData ([524](#))

### 20.7.5 TNamedIniCollection.IndexOfName

Synopsis: Search for an item, based on its name, and return its position

Declaration: `function IndexOfName(const AName: string) : Integer`

Visibility: public

Description: `IndexOfName` searches the list of items and returns the index of the item which has name equal to `AName` (case insensitive). If no such item exists, -1 is returned.

Note that the (linear) search starts at the last element and works it's way back to the first.

Errors: If no item exists, -1 is returned.

See also: IndexOfUserData ([522](#)), TNamedIniCollectionItem.Name ([524](#))

## 20.7.6 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 (522) 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 (522), FindByUserData (523)

## 20.7.7 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 (524) property value matches the UserData parameter. If no item is found, Nil is returned.

**Errors:** If no item is found, Nil is returned.

## 20.7.8 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 (523) items. The index is zero based.

**See also:** TNamedIniCollectionItem (523)

## 20.8 TNamedIniCollectionItem

### 20.8.1 Description

TNamedIniCollectionItem is a TIniCollectionItem (520) 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 (520)

### 20.8.2 Property overview

Page	Property	Access	Description
524	Name	rw	Name of the item
524	UserData	rw	User-defined data

### 20.8.3 **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`.

### 20.8.4 **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` ([524](#))

# Chapter 21

## Reference for unit 'IniFiles'

### 21.1 Used units

Table 21.1: Used units by unit 'IniFiles'

Name	Page
Classes	??
contnrs	<a href="#">119</a>
System	??
sysutils	??

### 21.2 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` ([525](#)) and is implemented in `TIniFile` ([536](#)) and `TMemIniFile` ([545](#)). The API presented by these components is Delphi compatible.

### 21.3 TCustomIniFile

#### 21.3.1 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` ([536](#)) or `TMemIniFile` ([545](#)).

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` ([536](#)), `TMemIniFile` ([545](#))

### 21.3.2 Method overview

Page	Property	Description
526	Create	Instantiate a new instance of TCustomIniFile.
534	DeleteKey	Delete a key from a section
527	Destroy	Remove the TCustomIniFile instance from memory
533	EraseSection	Clear a section
531	ReadBinaryStream	Read binary data
529	ReadBool	
530	ReadDate	Read a date value
530	ReadDateTime	Read a Date/Time value
530	ReadFloat	Read a floating point value
528	ReadInt64	Read Int64 value
528	ReadInteger	Read an integer value from the file
532	ReadSection	Read the key names in a section
533	ReadSections	Read the list of sections
533	ReadSectionValues	Read names and values of a section
527	ReadString	Read a string valued key
530	ReadTime	Read a time value
527	SectionExists	Check if a section exists.
534	UpdateFile	Update the file on disk
534	ValueExists	Check if a value exists
532	WriteBinaryStream	Write binary data
529	WriteBool	Write boolean value
531	WriteDate	Write date value
531	WriteDateTime	Write date/time value
532	WriteFloat	Write a floating-point value
529	WriteInt64	Write a Int64 value.
528	WriteInteger	Write an integer value
528	WriteString	Write a string value
532	WriteTime	Write time value

### 21.3.3 Property overview

Page	Property	Access	Description
535	CaseSensitive	rw	Are key and section names case sensitive
535	EscapeLineFeeds	r	Should linefeeds be escaped ?
534	FileName	r	Name of the .ini file
535	StripQuotes	rw	Should quotes be stripped from string values

### 21.3.4 TCustomIniFile.Create

**Synopsis:** Instantiate a new instance of TCustomIniFile.

**Declaration:** 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 AEscapeLineFeeds parameter is True, then lines which have their end-of-line markers escaped with a backslash, will be concatenated. This means that the following 2 lines

Description=This is a \

line with a long text

is equivalent to

```
Description=This is a line with a long text
```

By default, not escaping of linefeeds is performed (for Delphi compatibility)

**Errors:** If the file cannot be read, an exception may be raised.

See also: [Destroy \(527\)](#)

### 21.3.5 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 \(525\)](#)

### 21.3.6 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 \(534\)](#)

### 21.3.7 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 \(528\)](#), [ReadInteger \(528\)](#), [ReadBool \(529\)](#), [ReadDate \(530\)](#), [ReadDateTime \(530\)](#), [ReadTime \(530\)](#), [ReadFloat \(530\)](#), [ReadBinaryStream \(531\)](#)

### 21.3.8 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 \(527\)](#), [WriteInteger \(528\)](#), [WriteBool \(529\)](#), [WriteDate \(531\)](#), [WriteDateTime \(531\)](#),  
[WriteTime \(532\)](#), [WriteFloat \(532\)](#), [WriteBinaryStream \(532\)](#)

### 21.3.9 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 \(528\)](#), [ReadString \(527\)](#), [ReadBool \(529\)](#), [ReadDate \(530\)](#), [ReadDateTime \(530\)](#), [ReadTime \(530\)](#), [ReadFloat \(530\)](#), [ReadBinaryStream \(531\)](#)

### 21.3.10 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 \(528\)](#), [WriteString \(528\)](#), [WriteBool \(529\)](#), [WriteDate \(531\)](#), [WriteDateTime \(531\)](#),  
[WriteTime \(532\)](#), [WriteFloat \(532\)](#), [WriteBinaryStream \(532\)](#)

### 21.3.11 TCustomIniFile.ReadInt64

**Synopsis:** Read Int64 value

**Declaration:** `function ReadInt64(const Section: string; const Ident: string;  
Default: Int64) : LongInt; 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 (525) only reads at most a 32-bit value.

See also: ReadInteger (525), WriteInt64 (525)

### 21.3.12 TCustomIniFile.WriteInt64

**Synopsis:** Write a 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: WriteInteger (525), ReadInt64 (525)

### 21.3.13 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 (529), ReadInteger (528), ReadString (527), ReadDate (530), ReadDateTime (530), ReadTime (530), ReadFloat (530), ReadBinaryStream (531)

### 21.3.14 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 (529), WriteInteger (528), WriteString (528), WriteDate (531), WriteDateTime (531), WriteTime (532), WriteFloat (532), WriteBinaryStream (532)

### 21.3.15 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 ([531](#)), ReadInteger ([528](#)), ReadBool ([529](#)), ReadString ([527](#)), ReadDateTime ([530](#)), ReadTime ([530](#)), ReadFloat ([530](#)), ReadBinaryStream ([531](#))

### 21.3.16 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 ([531](#)), ReadInteger ([528](#)), ReadBool ([529](#)), ReadDate ([530](#)), ReadString ([527](#)), ReadTime ([530](#)), ReadFloat ([530](#)), ReadBinaryStream ([531](#))

### 21.3.17 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 ([532](#)), ReadInteger ([528](#)), ReadBool ([529](#)), ReadDate ([530](#)), ReadDateTime ([530](#)), ReadTime ([530](#)), ReadString ([527](#)), ReadBinaryStream ([531](#))

### 21.3.18 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` (532), `ReadInteger` (528), `ReadBool` (529), `ReadDate` (530), `ReadDateTime` (530), `ReadString` (527), `ReadFloat` (530), `ReadBinaryStream` (531)

### 21.3.19 **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` (532), `ReadInteger` (528), `ReadBool` (529), `ReadDate` (530), `ReadDateTime` (530), `ReadTime` (530), `ReadFloat` (530), `ReadString` (527)

### 21.3.20 **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` (530), `WriteInteger` (528), `WriteBool` (529), `WriteString` (528), `WriteDateTime` (531), `WriteTime` (532), `WriteFloat` (532), `WriteBinaryStream` (532)

### 21.3.21 **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` (530), `WriteInteger` (528), `WriteBool` (529), `WriteDate` (531), `WriteString` (528), `WriteTime` (532), `WriteFloat` (532), `WriteBinaryStream` (532)

### 21.3.22 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 \(530\)](#), [WriteInteger \(528\)](#), [WriteBool \(529\)](#), [WriteDate \(531\)](#), [WriteDateTime \(531\)](#), [WriteTime \(532\)](#), [WriteString \(528\)](#), [WriteBinaryStream \(532\)](#)

### 21.3.23 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 \(530\)](#), [WriteInteger \(528\)](#), [WriteBool \(529\)](#), [WriteDate \(531\)](#), [WriteDateTime \(531\)](#), [WriteString \(528\)](#), [WriteFloat \(532\)](#), [WriteBinaryStream \(532\)](#)

### 21.3.24 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 \(531\)](#), [WriteInteger \(528\)](#), [WriteBool \(529\)](#), [WriteDate \(531\)](#), [WriteDateTime \(531\)](#), [WriteTime \(532\)](#), [WriteFloat \(532\)](#), [WriteString \(528\)](#)

### 21.3.25 TCustomIniFile.ReadSection

**Synopsis:** Read the key names in a section

**Declaration:** `procedure ReadSection(const Section: string; Strings: TStrings); Virtual  
; Abstract`

Visibility: public

Description: ReadSection will return the names of the keys in section *Section* in *Strings*, one string per key. If a non-existing section is specified, the list is cleared. To return the values of the keys as well, the ReadSectionValues ([533](#)) method should be used.

See also: [ReadSections](#) ([533](#)), [SectionExists](#) ([527](#)), [ReadSectionValues](#) ([533](#))

### **21.3.26 TCustomIniFile.ReadSections**

Synopsis: Read the list of sections

Declaration: procedure ReadSections(*Strings*: TStrings); Virtual; Abstract

Visibility: public

Description: ReadSections returns the names of existing sections in *Strings*. It also returns names of empty sections.

See also: [SectionExists](#) ([527](#)), [ReadSectionValues](#) ([533](#)), [ReadSection](#) ([532](#))

### **21.3.27 TCustomIniFile.ReadSectionValues**

Synopsis: Read names and values of a section

Declaration: procedure ReadSectionValues(const *Section*: string; *Strings*: TStrings)  
; Virtual; Abstract

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 ([532](#)) can be used.

See also: [SectionExists](#) ([527](#)), [ReadSections](#) ([533](#)), [ReadSection](#) ([532](#))

### **21.3.28 TCustomIniFile.EraseSection**

Synopsis: Clear a section

Declaration: procedure EraseSection(const *Section*: string); Virtual; Abstract

Visibility: public

Description: EraseSection deletes all values from the section named *Section* and removes the section from the ini file. If the section didn't exist prior to a call to EraseSection, nothing happens.

See also: [SectionExists](#) ([527](#)), [ReadSections](#) ([533](#)), [DeleteKey](#) ([534](#))

### **21.3.29 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 ([533](#))

### **21.3.30 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 immediatly 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 ([540](#))

### **21.3.31 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 ([527](#))

### **21.3.32 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 ([526](#))

### **21.3.33 TCustomIniFile.EscapeLineFeeds**

**Synopsis:** Should linefeeds be escaped ?

**Declaration:** Property EscapeLineFeeds : Boolean

**Visibility:** public

**Access:** Read

**Description:** EscapeLineFeeds determines whether escaping of linefeeds is enabled: For a description of this feature, see Create ([526](#)), as the value of this property must be specified when the TCustomIniFile instance is created.

By default, EscapeLineFeeds is False.

**See also:** Create ([526](#)), CaseSensitive ([535](#))

### **21.3.34 TCustomIniFile.CaseSensitive**

**Synopsis:** Are key and section names case sensitive

**Declaration:** Property CaseSensitive : Boolean

**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 ([535](#))

### **21.3.35 TCustomIniFile.StripQuotes**

**Synopsis:** Should quotes be stripped from string values

**Declaration:** Property StripQuotes : Boolean

**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).

## **21.4 THashedStringList**

### **21.4.1 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 ([536](#)) and IndexOfName ([536](#)) functions make use of these hash values to quicklier locate a value.

**See also:** IndexOf ([536](#)), IndexOfName ([536](#))

### 21.4.2 Method overview

Page	Property	Description
<a href="#">536</a>	Destroy	Clean up instance
<a href="#">536</a>	IndexOf	Returns the index of a string in the list of strings
<a href="#">536</a>	IndexOfName	Return the index of a name in the list of name=value pairs

### 21.4.3 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`.

See also: `THashedStringList.Create` ([535](#))

### 21.4.4 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` ([536](#))

### 21.4.5 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` ([536](#))

## 21.5 TIniFile

### 21.5.1 Description

`TIniFile` is an implementation of `TCustomIniFile` ([525](#)) which does the same as `TMemIniFile` ([545](#)), 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` ([525](#)), `TMemIniFile` ([545](#))

### 21.5.2 Method overview

Page	Property	Description
<a href="#">537</a>	Create	Create a new instance of <code>TIniFile</code>
<a href="#">539</a>	DeleteKey	Delete key
<a href="#">537</a>	Destroy	Remove the <code>TIniFile</code> instance from memory
<a href="#">539</a>	EraseSection	
<a href="#">538</a>	ReadSection	Read the key names in a section
<a href="#">538</a>	ReadSectionRaw	Read raw section
<a href="#">539</a>	ReadSections	Read section names
<a href="#">539</a>	ReadSectionValues	
<a href="#">538</a>	ReadString	Read a string
<a href="#">540</a>	UpdateFile	Update the file on disk
<a href="#">538</a>	WriteString	Write string to file

### 21.5.3 Property overview

Page	Property	Access	Description
<a href="#">540</a>	CacheUpdates	rw	Should changes be kept in memory
<a href="#">540</a>	Stream	r	Stream from which ini file was read

### 21.5.4 `TIniFile.Create`

**Synopsis:** Create a new instance of `TIniFile`

**Declaration:** constructor Create(const AFileName: string; AEscapeLineFeeds: Boolean);  
constructor Create(AStream: TStream; AEscapeLineFeeds: Boolean)

**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`.

**See also:** `TCustomIniFile` ([525](#)), `TMemIniFile` ([545](#))

### 21.5.5 `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` ([534](#)), `CacheUpdates` ([540](#))

### **21.5.6 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 ([527](#)) abstract method by looking at the in-memory copy of the ini file and returning the string found there.

**See also:** TCustomIniFile.ReadString ([527](#))

### **21.5.7 TIniFile.WriteString**

**Synopsis:** Write string to file

**Declaration:** procedure WriteString(const Section: string;const Ident: string;  
const Value: string);   Override

**Visibility:** public

**Description:** WriteString implements the TCustomIniFile.WriteString ([528](#)) abstract method by writing the string to the in-memory copy of the ini file. If CacheUpdates ([540](#)) 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.

### **21.5.8 TIniFile.ReadSection**

**Synopsis:** Read the key names in a section

**Declaration:** procedure ReadSection(const Section: string;Strings: TStrings)  
;   Override

**Visibility:** public

**Description:** ReadSection reads the key names from Section into Strings, taking the in-memory copy of the ini file. This is the implementation for the abstract TCustomIniFile.ReadSection ([532](#))

**See also:** TCustomIniFile.ReadSection ([532](#)), TIniFile.ReadSectionRaw ([538](#))

### **21.5.9 TIniFile.ReadSectionRaw**

**Synopsis:** Read raw section

**Declaration:** procedure ReadSectionRaw(const Section: string;Strings: TStrings)

**Visibility:** public

**Description:** ReadSectionRaw returns the contents of the section Section as it is: this includes the comments in the section. (these are also stored in memory)

**See also:** TIniFile.ReadSection ([538](#)), TCustomIniFile.ReadSection ([532](#))

### 21.5.10 TIniFile.ReadSections

**Synopsis:** Read section names

**Declaration:** procedure ReadSections(Strings: TStrings); Override

**Visibility:** public

**Description:** ReadSections is the implementation of TCustomIniFile.ReadSections (533). It operates on the in-memory copy of the ini file, and places all section names in Strings.

**See also:** TIniFile.ReadSection (538), TCustomIniFile.ReadSections (533), TIniFile.ReadSectionValues (539)

### 21.5.11 TIniFile.ReadSectionValues

**Synopsis:**

**Declaration:** procedure ReadSectionValues(const Section: string; Strings: TStrings); Override

**Visibility:** public

**Description:** ReadSectionValues is the implementation of TCustomIniFile.ReadSectionValues (533). It operates on the in-memory copy of the ini file, and places all key names from Section together with their values in Strings.

**See also:** TIniFile.ReadSection (538), TCustomIniFile.ReadSectionValues (533), TIniFile.ReadSections (539)

### 21.5.12 TIniFile.EraseSection

**Synopsis:**

**Declaration:** procedure EraseSection(const Section: string); Override

**Visibility:** public

**Description:** EraseSection deletes the section Section from memory, if CacheUpdates (540) is False, then the file is immediately updated on disk. This method is the implementation of the abstract TCustomIniFile.EraseSection (533) method.

**See also:** TCustomIniFile.EraseSection (533), TIniFile.ReadSection (538), TIniFile.ReadSections (539)

### 21.5.13 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 (540) is False, then the file is immediately updated on disk.

**See also:** CacheUpdates (540)

### 21.5.14 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 ([540](#))

### 21.5.15 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 ([540](#)) method will use this stream to write changes to.

See also: Create ([537](#)), UpdateFile ([540](#))

### 21.5.16 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 ([540](#)). 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 ([540](#))

## 21.6 TIniFileKey

### 21.6.1 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 ([536](#)) class.

See also: TIniFile ([536](#))

## 21.6.2 Method overview

Page	Property	Description
<a href="#">541</a>	Create	Create a new instance of TIniFileKey

## 21.6.3 Property overview

Page	Property	Access	Description
<a href="#">541</a>	Ident	rw	Key name
<a href="#">541</a>	Value	rw	Key value

### 21.6.4 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 ([541](#)) with AIdent and Value ([541](#)) with AValue.

See also: Ident ([541](#)), Value ([541](#))

### 21.6.5 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 ([541](#))

### 21.6.6 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 ([541](#))

## 21.7 TIniFileKeyList

### 21.7.1 Description

TIniFileKeyList maintains a list of TIniFileKey (540) instances on behalf of the TIniFileSection (543) class. It stores they keys of one section of the .ini files.

See also: TIniFileKey (540), TIniFileSection (543)

### 21.7.2 Method overview

Page	Property	Description
542	Clear	Clear the list
542	Destroy	Free the instance

### 21.7.3 Property overview

Page	Property	Access	Description
542	Items	r	Indexed access to TIniFileKey items in the list

### 21.7.4 TIniFileKeyList.Destroy

Synopsis: Free the instance

Declaration: destructor Destroy; Override

Visibility: public

Description: Destroy clears up the list using Clear (542) and then calls the inherited destroy.

See also: Clear (542)

### 21.7.5 TIniFileKeyList.Clear

Synopsis: Clear the list

Declaration: procedure Clear; Override

Visibility: public

Description: Clear removes all TIniFileKey (540) instances from the list, and frees the instances.

See also: TIniFileKey (540)

### 21.7.6 TIniFileKeyList.Items

Synopsis: Indexed access to TIniFileKey items in the list

Declaration: Property Items[Index: Integer]: TIniFileKey; default

Visibility: public

Access: Read

Description: Items provides indexed access to the TIniFileKey (540) items in the list. The index is zero-based and runs from 0 to Count-1.

See also: TIniFileKey (540)

## 21.8 TIniFileSection

### 21.8.1 Description

TIniFileSection is a class which represents a section in the .ini, and is used internally by the TIniFile (536) class (one instance of TIniFileSection is created for each section in the file by the TIniFileSectionList (544) list). The name of the section is stored in the Name (544) property, and the key/value pairs in this section are available in the KeyList (544) property.

See also: TIniFileKeyList (542), TIniFile (536), TIniFileSectionList (544)

### 21.8.2 Method overview

Page	Property	Description
543	Create	Create a new section object
543	Destroy	Free the section object from memory
543	Empty	Is the section empty

### 21.8.3 Property overview

Page	Property	Access	Description
544	KeyList	r	List of key/value pairs in this section
544	Name	r	Name of the section

### 21.8.4 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.

### 21.8.5 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 (542) instance to keep all the key/value pairs for this section.

See also: TIniFileKeyList (542)

### 21.8.6 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 ([543](#)), TIniFileKeyList ([542](#))

### 21.8.7 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 ([544](#))

### 21.8.8 TIniFileSection.KeyList

**Synopsis:** List of key/value pairs in this section

**Declaration:** Property KeyList : TIniFileKeyList

**Visibility:** public

**Access:** Read

**Description:** KeyList is the TIniFileKeyList ([542](#)) instance that is used by the TIniFileSection to keep the key/value pairs of the section.

See also: TIniFileSection.Name ([544](#)), TIniFileKeyList ([542](#))

## 21.9 TIniFileSectionList

### 21.9.1 Description

TIniFileSectionList maintains a list of TIniFileSection ([543](#)) instances, one for each section in an .ini file. TIniFileSectionList is used internally by the TIniFile ([536](#)) class to represent the sections in the file.

See also: TIniFileSection ([543](#)), TIniFile ([536](#))

### 21.9.2 Method overview

Page	Property	Description
<a href="#">545</a>	Clear	Clear the list
<a href="#">545</a>	Destroy	Free the object from memory

### 21.9.3 Property overview

Page	Property	Access	Description
<a href="#">545</a>	Items	r	Indexed access to all the section objects in the list

### 21.9.4 TIniFileSectionList.Destroy

**Synopsis:** Free the object from memory

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

**Visibility:** public

**Description:** `Destroy` calls `Clear` (545) to clear the section list and then calls the inherited `Destroy`

**See also:** `Clear` (545)

### 21.9.5 TIniFileSectionList.Clear

**Synopsis:** Clear the list

**Declaration:** `procedure Clear; Override`

**Visibility:** public

**Description:** `Clear` removes all `TIniFileSection` (543) items from the list, and frees the items it removes from the list.

**See also:** `TIniFileSection` (543), `TIniFileSectionList.Items` (545)

### 21.9.6 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` (543), `TIniFileSectionList.Clear` (545)

## 21.10 TMemIniFile

### 21.10.1 Description

`TMemIniFile` is a simple descendent of `TIniFile` (536) 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` (536), `TCustomIniFile` (525), `CacheUpdates` (540)

### 21.10.2 Method overview

Page	Property	Description
<a href="#">546</a>	<code>Clear</code>	Clear the data
<a href="#">546</a>	<code>Create</code>	Create a new instance of <code>TMemIniFile</code>
<a href="#">546</a>	<code>GetStrings</code>	Get contents of ini file as stringlist
<a href="#">546</a>	<code>Rename</code>	Rename the ini file
<a href="#">547</a>	<code>SetStrings</code>	Set data from a stringlist

### **21.10.3 TMemIniFile.Create**

**Synopsis:** Create a new instance of TMemIniFile

**Declaration:** constructor Create(const AFileName: string; AEscapeLineFeeds: Boolean)  
;   Override

**Visibility:** public

**Description:** Create simply calls the inherited Create (537), and sets the CacheUpdates (540) to True so updates will be kept in memory till they are explicitly written to disk.

**See also:** TIniFile.Create (537), CacheUpdates (540)

### **21.10.4 TMemIniFile.Clear**

**Synopsis:** Clear the data

**Declaration:** procedure Clear

**Visibility:** public

**Description:** Clear removes all sections and key/value pairs from memory. If CacheUpdates (540) is set to False then the file on disk will immediatly be emptied.

**See also:** SetStrings (547), GetStrings (546)

### **21.10.5 TMemIniFile.GetStrings**

**Synopsis:** Get contents of ini file as stringlist

**Declaration:** procedure GetStrings(List: TStrings)

**Visibility:** public

**Description:** GetStrings returns the whole contents of the ini file in a single stringlist, List. This includes comments and empty sections.

The GetStrings call can be used to get data for a call to SetStrings (547), which can be used to copy data between 2 in-memory ini files.

**See also:** SetStrings (547), Clear (546)

### **21.10.6 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 (540)

### **21.10.7 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 (546). The two calls combined can be used to copy data between 2 in-memory ini files.

**See also:** GetStrings (546), Clear (546)

# Chapter 22

## Reference for unit 'iostream'

### 22.1 Used units

Table 22.1: Used units by unit 'iostream'

Name	Page
Classes	??
System	??

### 22.2 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`).

### 22.3 Constants, types and variables

#### 22.3.1 Types

```
TIOSType = (iosInput,iosOutPut,iosError)
```

Table 22.2: Enumeration values for type TIOSType

Value	Explanation
iosError	The stream can be used to write to standard diagnostic output
iosInput	The stream can be used to read from standard input
iosOutPut	The stream can be used to write to standard output

`TIOSType` is passed to the `Create` (549) constructor of `TIOStream` (549), it determines what kind of stream is created.

## 22.4 EIOStreamError

### 22.4.1 Description

Error thrown in case of an invalid operation on a TIOStream ([549](#)).

## 22.5 TIOStream

### 22.5.1 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 ([548](#)) 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 ([550](#)) behaviour based on this position.

See also: TIOSType ([548](#)), THandleStream ([??](#))

### 22.5.2 Method overview

Page	Property	Description
<a href="#">549</a>	Create	Construct a new instance of TIOStream ( <a href="#">549</a> )
<a href="#">549</a>	Read	Read data from the stream.
<a href="#">550</a>	Seek	Set the stream position
<a href="#">550</a>	Write	Write data to the stream

### 22.5.3 TIOStream.Create

Synopsis: Construct a new instance of TIOStream ([549](#))

Declaration: constructor Create(aIOSType: TIOSType)

Visibility: public

Description: Create creates a new instance of TIOStream ([549](#)), 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 ([549](#)), TIOStream.Write ([550](#))

### 22.5.4 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 ([549](#)) 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:** `TIOStreamType` (548), `TIOStream.Write` (550)

### 22.5.5 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` (549) 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:** `TIOStreamType` (548), `TIOStream.Read` (549)

### 22.5.6 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` (549) exception is raised if the stream does not allow the requested seek operation.

**See also:** `EIOStreamError` (549)

# Chapter 23

## Reference for unit 'libtar'

### 23.1 Used units

Table 23.1: Used units by unit 'libtar'

Name	Page
BaseUnix	??
Classes	??
System	??
sysutils	??
Unix	??
UnixType	??
Windows	??

### 23.2 Overview

The `libtar` units provides 2 classes to read and write `.tar` archives: `TTarArchive` (555) class can be used to read a tar file, and the `TTarWriter` (557) class can be used to write a tar file. The unit was implemented originally by Stefan Heymann.

### 23.3 Constants, types and variables

#### 23.3.1 Constants

```
ALL_PERMISSIONS = [tpReadByOwner, tpWriteByOwner, tpExecuteByOwner, tpReadByGroup, tp
```

`ALL_PERMISSIONS` is a set constant containing all possible permissions (read/write/execute, for all groups of users) for an archive entry.

```
EXECUTE_PERMISSIONS = [tpExecuteByOwner, tpExecuteByGroup, tpExecuteByOther]
```

`WRITE_PERMISSIONS` is a set constant containing all possible execute permissions set for an archive entry.

```
FILETYPE_NAME : Array[TF fileType] of string = ('Regular', 'Link', 'Symbolic Link', 'C
```

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.

### 23.3.2 Types

```
TF fileType = (ftNormal, ftLink, ftSymbolicLink, ftCharacter, ftBlock,
               ftDirectory, ftFifo, ftContiguous, ftDumpDir, ftMultiVolume,
               ftVolumeHeader)
```

Table 23.2: Enumeration values for type TF fileType

Value	Explanation
ftBlock	Block device file
ftCharacter	Character device file
ftContiguous	Contiguous file
ftDirectory	Directory
ftDumpDir	List of files
ftFifo	FIFO file
ftLink	Hard link
ftMultiVolume	Multi-volume file part
ftNormal	Normal file
ftSymbolicLink	Symbolic link
ftVolumeHeader	Volume header, can appear only as first entry in the archive

TF fileType describes the file type of a file in the archive. It is used in the FileType field of the TTarDirRec ([553](#)) record.

```
TTarDirRec = record
  Name : AnsiString;
  Size : Int64;
  DateTime : TDateTime;
  Permissions : TTarPermissions;
  FileType : TF fileType;
  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` (556) call.

```
TTarMode = (tmSetUid,tmSetGid,tmSaveText)
```

Table 23.3: Enumeration values for type `TTarMode`

Value	Explanation
tmSaveText	Bit \$200 is set
tmSetGid	File has SetGID bit set
tmSetUid	File has SetUID bit set.

`TTarMode` describes extra file modes. It is used in the `Mode` field of the `TTarDirRec` (553) record.

```
TTarModes = Set of TTarMode
```

`TTarModes` denotes the full set of permission bits for the file in the `Mode` field of the `TTarDirRec` (553) record.

```
TTarPermission = (tpReadByOwner,tpWriteByOwner,tpExecuteByOwner,
                  tpReadByGroup,tpWriteByGroup,tpExecuteByGroup,
                  tpReadByOther,tpWriteByOther,tpExecuteByOther)
```

Table 23.4: Enumeration values for type `TTarPermission`

Value	Explanation
tpExecuteByGroup	Group can execute the file
tpExecuteByOther	Other people can execute the file
tpExecuteByOwner	Owner can execute the file
tpReadByGroup	Group can read the file
tpReadByOther	Other people can read the file.
tpReadByOwner	Owner can read the file
tpWriteByGroup	Group can write the file
tpWriteByOther	Other people can write the file
tpWriteByOwner	Owner can write the file

`TTarPermission` denotes part of a files permission as it is 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` (553) record.

## 23.4 Procedures and functions

### 23.4.1 ClearDirRec

Synopsis: Initialize tar archive entry

Declaration: procedure ClearDirRec(var DirRec: TTarDirRec)

Visibility: default

Description: ClearDirRec clears the DirRec entry, it basically zeroes out all fields.

See also: TTarDirRec ([553](#))

### 23.4.2 ConvertFilename

Synopsis: Convert filename to archive format

Declaration: function ConvertFilename(Filename: string) : string

Visibility: default

Description: ConvertFileName converts the file name FileName to a format allowed by the tar archive.  
Basically, it converts directory specifiers to forward slashes.

### 23.4.3 FileTimeGMT

Synopsis: Extract filetime

Declaration: function FileTimeGMT(FileName: string) : TDateTime; Overload  
function FileTimeGMT(SearchRec: TSearchRec) : TDateTime; Overload

Visibility: default

Description: FileTimeGMT returns the timestamp of a filename (FileName must exist) or a search rec (TSearchRec) to a GMT representation that can be used in a tar entry.

See also: TTarDirRec ([553](#))

### 23.4.4 PermissionString

Synopsis: Convert a set of permissions to a string

Declaration: function PermissionString(Permissions: TTarPermissions) : string

Visibility: default

Description: PermissionString can be used to convert a set of Permissions to a string in the same format as used by the unix 'ls' command.

See also: TTarPermissions ([553](#))

## 23.5 TTarArchive

### 23.5.1 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 (557), FindNext (556)

### 23.5.2 Method overview

Page	Property	Description
<a href="#">555</a>	Create	Create a new instance of the archive
<a href="#">555</a>	Destroy	Destroy TTarArchive instance
<a href="#">556</a>	FindNext	Find next archive entry
<a href="#">556</a>	GetFilePos	Return current archive position
<a href="#">556</a>	ReadFile	Read a file from the archive
<a href="#">555</a>	Reset	Reset archive
<a href="#">557</a>	SetFilePos	Set position in archive

### 23.5.3 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 StreamTStream (??) 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 (556)

### 23.5.4 TTarArchive.Destroy

Synopsis: Destroy TTarArchive instance

Declaration: destructor Destroy; Override

Visibility: public

Description: Destroy closes the archive stream (if it created a stream) and cleans up the TTarArchive instance.

See also: TTarArchive.Create (555)

### 23.5.5 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 ([555](#))

### 23.5.6 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 ([556](#))

### 23.5.7 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.

### 23.5.8 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 ([557](#)), TTarArchive.Reset ([555](#))

### 23.5.9 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 ([555](#)), TTarArchive.GetFilePos ([556](#))

## 23.6 TTarWriter

### 23.6.1 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 ([555](#))

### 23.6.2 Method overview

Page	Property	Description
<a href="#">559</a>	AddDir	Add directory to archive
<a href="#">558</a>	AddFile	Add a file to the archive
<a href="#">560</a>	AddLink	Add hard link to archive
<a href="#">558</a>	AddStream	Add stream contents to archive.
<a href="#">559</a>	AddString	Add string as file data
<a href="#">559</a>	AddSymbolicLink	Add a symbolic link to the archive
<a href="#">560</a>	AddVolumeHeader	Add volume header entry
<a href="#">557</a>	Create	Create a new archive
<a href="#">558</a>	Destroy	Close archive and clean up TTarWriter
<a href="#">560</a>	Finalize	Finalize the archive

### 23.6.3 Property overview

Page	Property	Access	Description
<a href="#">561</a>	GID	rw	Archive entry group ID
<a href="#">561</a>	GroupName	rw	Archive entry group name
<a href="#">562</a>	Magic	rw	Archive entry Magic constant
<a href="#">562</a>	Mode	rw	Archive entry mode
<a href="#">560</a>	Permissions	rw	Archive entry permissions
<a href="#">561</a>	UID	rw	Archive entry user ID
<a href="#">561</a>	UserName	rw	Archive entry user name

### 23.6.4 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 ([558](#))

### 23.6.5 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 ([560](#))

### 23.6.6 TTarWriter.AddFile

Synopsis: Add a file to the archive

Declaration: procedure AddFile(Filename: string; TarFilename: AnsiString)

Visibility: public

Description: AddFile adds a file to the archive: the contents is read from FileName. Optionally, an alternative filename can be specified in TarFileName. This name should contain only forward slash path separators. If it is not specified, the name will be computed from FileName.

The archive entry is written with the current owner data and permissions.

Errors: If FileName cannot be opened, an exception will be raised.

See also: TTarWriter.AddStream ([558](#)), TTarWriter.AddString ([559](#)), TTarWriter.AddLink ([560](#)), TTarWriter.AddSymbolicLink ([559](#)), TTarWriter.AddDir ([559](#)), TTarWriter.AddVolumeHeader ([560](#))

### 23.6.7 TTarWriter.AddStream

Synopsis: Add stream contents to archive.

Declaration: procedure AddStream(Stream: TStream; TarFilename: AnsiString; FileDateGmt: TDateTime)

Visibility: public

Description: AddStream will add the contents of Stream to the archive. The Stream will not be reset: only the contents of the stream from the current position will be written to the archive. The entry will be written with file name TarFileName. This name should contain only forward slash path separators. The entry will be written with timestamp FileDateGmt.

The archive entry is written with the current owner data and permissions.

See also: TTarWriter.AddFile ([558](#)), TTarWriter.AddString ([559](#)), TTarWriter.AddLink ([560](#)), TTarWriter.AddSymbolicLink ([559](#)), TTarWriter.AddDir ([559](#)), TTarWriter.AddVolumeHeader ([560](#))

### **23.6.8 TTarWriter.AddString**

**Synopsis:** Add string as file data

**Declaration:** procedure AddString(Contents: Ansistring; TarFilename: AnsiString;  
FileDateGmt: TDateTime)

**Visibility:** public

**Description:** AddString adds the string `Contents` as the data of an entry with file name `TarFileName`.  
This name should contain only forward slash path separators. The entry will be written with timestamp `FileDateGmt`.

The archive entry is written with the current owner data and permissions.

**See also:** TTarWriter.AddFile ([558](#)), TTarWriter.AddStream ([558](#)), TTarWriter.AddLink ([560](#)), TTarWriter.AddSymbolicLink ([559](#)), TTarWriter.AddDir ([559](#)), TTarWriter.AddVolumeHeader ([560](#))

### **23.6.9 TTarWriter.AddDir**

**Synopsis:** Add directory to archive

**Declaration:** procedure AddDir(Dirname: AnsiString; DateGmt: TDateTime;  
MaxDirSize: Int64)

**Visibility:** public

**Description:** AddDir adds a directory entry to the archive. The entry is written with name `DirName`, maximum directory size `MaxDirSize` (0 means unlimited) and timestamp `DateGmt`.

Note that this call only adds an entry for a directory to the archive: if `DirName` is an existing directory, it does not write all files in the directory to the archive.

The directory entry is written with the current owner data and permissions.

**See also:** TTarWriter.AddFile ([558](#)), TTarWriter.AddStream ([558](#)), TTarWriter.AddLink ([560](#)), TTarWriter.AddSymbolicLink ([559](#)), TTarWriter.AddString ([559](#)), TTarWriter.AddVolumeHeader ([560](#))

### **23.6.10 TTarWriter.AddSymbolicLink**

**Synopsis:** Add a symbolic link to the archive

**Declaration:** procedure AddSymbolicLink(Filename: AnsiString; Linkname: AnsiString;  
DateGmt: TDateTime)

**Visibility:** public

**Description:** AddSymbolicLink adds a symbolic link entry to the archive, with name `FileName`, pointing to `LinkName`. The entry is written with timestamp `DateGmt`.

The link entry is written with the current owner data and permissions.

**See also:** TTarWriter.AddFile ([558](#)), TTarWriter.AddStream ([558](#)), TTarWriter.AddLink ([560](#)), TTarWriter.AddDir ([559](#)), TTarWriter.AddString ([559](#)), TTarWriter.AddVolumeHeader ([560](#))

### **23.6.11 TTarWriter.AddLink**

**Synopsis:** Add hard link to archive

**Declaration:** procedure AddLink(Filename: AnsiString; Linkname: AnsiString;  
DateGmt: TDateTime)

**Visibility:** public

**Description:** AddLink adds a hard link entry to the archive. The entry has name FileName, timestamp DateGmt and points to LinkName.

The link entry is written with the current owner data and permissions.

**See also:** TTarWriter.AddFile ([558](#)), TTarWriter.AddStream ([558](#)), TTarWriter.AddSymbolicLink ([559](#)), TTarWriter.AddDir ([559](#)), TTarWriter.AddString ([559](#)), TTarWriter.AddVolumeHeader ([560](#))

### **23.6.12 TTarWriter.AddVolumeHeader**

**Synopsis:** Add volume header entry

**Declaration:** procedure AddVolumeHeader(VolumeId: AnsiString; DateGmt: TDateTime)

**Visibility:** public

**Description:** AddVolumeHeader adds a volume header entry to the archive. The entry is written with name VolumeID and timestamp DateGmt.

The volume header entry is written with the current owner data and permissions.

**See also:** TTarWriter.AddFile ([558](#)), TTarWriter.AddStream ([558](#)), TTarWriter.AddSymbolicLink ([559](#)), TTarWriter.AddDir ([559](#)), TTarWriter.AddString ([559](#)), TTarWriter.AddLink ([560](#))

### **23.6.13 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 ([558](#))

### **23.6.14 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 ([553](#))

### 23.6.15 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 ([553](#))

### 23.6.16 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 ([553](#))

### 23.6.17 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 ([553](#))

### 23.6.18 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 ([553](#))

### **23.6.19 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 ([553](#))

### **23.6.20 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 ([553](#))

# Chapter 24

## Reference for unit 'mssqlconn'

### 24.1 Used units

Table 24.1: Used units by unit 'mssqlconn'

Name	Page
BufDataset	??
Classes	??
db	<a href="#">231</a>
dblib	??
sqldb	<a href="#">630</a>
System	??
sysutils	??

### 24.2 Overview

Connector to Microsoft SQL Server databases. Needs FreeTDS dblib library.

### 24.3 Constants, types and variables

#### 24.3.1 Types

```
TClientCharset = (ccNone, ccUTF8, ccISO88591, ccUnknown)
```

Table 24.2: Enumeration values for type TClientCharset

Value	Explanation
ccISO88591	
ccNone	
ccUnknown	
ccUTF8	

```
TServerInfo = record
  ServerVersion : string;
  ServerVersionString : string;
  UserName : string;
end
```

### 24.3.2 Variables

```
DBLibLibraryName : string = DBLIBDLL
```

## 24.4 EMSSQLDatabaseError

### 24.4.1 Description

Sybase/MS SQL Server specific error

## 24.5 TMSSQLConnection

### 24.5.1 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 TMSSQLConnection.Params: 'TEXTSIZE=2147483647' or execute 'SET TEXTSIZE 2147483647')

### 24.5.2 Method overview

Page	Property	Description
<a href="#">565</a>	Create	
<a href="#">565</a>	CreateDB	
<a href="#">565</a>	DropDB	
<a href="#">565</a>	GetConnectionInfo	

### 24.5.3 Property overview

Page	Property	Access	Description
<a href="#">566</a>	CharSet		
<a href="#">567</a>	Connected		
<a href="#">567</a>	DatabaseName		
<a href="#">566</a>	HostName		Host and optionally port or instance
<a href="#">567</a>	KeepConnection		
<a href="#">567</a>	LoginPrompt		
<a href="#">568</a>	OnLogin		
<a href="#">567</a>	Params		
<a href="#">565</a>	Password		
<a href="#">567</a>	Role		
<a href="#">566</a>	Transaction		
<a href="#">566</a>	UserName		

### 24.5.4 TMSSQLConnection.Create

Declaration: constructor Create(AOwner: TComponent); Override

Visibility: public

### 24.5.5 TMSSQLConnection.GetConnectionInfo

Declaration: function GetConnectionInfo(InfoType: TConnInfoType) : string; Override

Visibility: public

### 24.5.6 TMSSQLConnection.CreateDB

Declaration: procedure CreateDB; Override

Visibility: public

### 24.5.7 TMSSQLConnection.DropDB

Declaration: procedure DropDB; Override

Visibility: public

### 24.5.8 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).

### 24.5.9 TMSSQLConnection.Transaction

Declaration: Property Transaction :

Visibility: published

Access:

### 24.5.10 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).

### 24.5.11 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.

### 24.5.12 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>

### **24.5.13 TMSSQLConnection.Connected**

Declaration: Property Connected :

Visibility: published

Access:

### **24.5.14 TMSSQLConnection.Role**

Declaration: Property Role :

Visibility: published

Access:

### **24.5.15 TMSSQLConnection.DatabaseName**

Declaration: Property DatabaseName :

Visibility: published

Access:

Description: TMSSQLConnection specific: the master database should always exist on a server.

### **24.5.16 TMSSQLConnection.KeepConnection**

Declaration: Property KeepConnection :

Visibility: published

Access:

### **24.5.17 TMSSQLConnection.LoginPrompt**

Declaration: Property LoginPrompt :

Visibility: published

Access:

### **24.5.18 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.

### **24.5.19 TMSSQLConnection.OnLogin**

Declaration: Property OnLogin :

Visibility: published

Access:

## **24.6 TMSSQLConnectionDef**

### **24.6.1 Method overview**

Page	Property	Description
568	ConnectionClass	
568	DefaultLibraryName	
568	Description	
569	LoadedLibraryName	
568	LoadFunction	
568	TypeName	
569	UnLoadFunction	

### **24.6.2 TMSSQLConnectionDef.TypeName**

Declaration: class function TypeName;   Override

Visibility: default

### **24.6.3 TMSSQLConnectionDef.ConnectionClass**

Declaration: class function ConnectionClass;   Override

Visibility: default

### **24.6.4 TMSSQLConnectionDef.Description**

Declaration: class function Description;   Override

Visibility: default

### **24.6.5 TMSSQLConnectionDef.DefaultLibraryName**

Declaration: class function DefaultLibraryName;   Override

Visibility: default

### **24.6.6 TMSSQLConnectionDef.LoadFunction**

Declaration: class function LoadFunction;   Override

Visibility: default

### **24.6.7 TMSSQLConnectionDef.UnLoadFunction**

Declaration: class function UnLoadFunction;   Override

Visibility: default

### **24.6.8 TMSSQLConnectionDef.LoadedLibraryName**

Declaration: class function LoadedLibraryName;   Override

Visibility: default

## **24.7 TSybaseConnection**

### **24.7.1 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)

### **24.7.2 Method overview**

Page	Property	Description
<a href="#">569</a>	Create	

### **24.7.3 TSybaseConnection.Create**

Declaration: constructor Create(AOwner: TComponent);   Override

Visibility: public

## **24.8 TSybaseConnectionDef**

### **24.8.1 Method overview**

Page	Property	Description
<a href="#">570</a>	ConnectionClass	
<a href="#">570</a>	Description	
<a href="#">570</a>	TypeName	

#### **24.8.2 TSybaseConnectionDef.TypeName**

Declaration: class function TypeName;   Override

Visibility: default

#### **24.8.3 TSybaseConnectionDef.ConnectionClass**

Declaration: class function ConnectionClass;   Override

Visibility: default

#### **24.8.4 TSybaseConnectionDef.Description**

Declaration: class function Description;   Override

Visibility: default

# Chapter 25

## Reference for unit 'Pipes'

### 25.1 Used units

Table 25.1: Used units by unit 'Pipes'

Name	Page
Classes	??
System	??
sysutils	??

### 25.2 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.

### 25.3 Constants, types and variables

#### 25.3.1 Constants

ENoSeekMsg = 'Cannot seek on pipes'

Constant used in EPipeSeek ([572](#)) exception.

EPipeMsg = 'Failed to create pipe.'

Constant used in EPipeCreation ([572](#)) exception.

### 25.4 Procedures and functions

#### 25.4.1 CreatePipeHandles

Synopsis: Function to create a set of pipe handles

**Declaration:** function CreatePipeHandles(var InHandle: THandle; var OutHandle: THandle; APipeBufferSize: Cardinal) : Boolean

**Visibility:** default

**Description:** CreatePipeHandles provides an OS-independent way to create a set of pipe filehandles. These handles are inheritable to child processes. The reading end of the pipe is returned in InHandle, the writing end in OutHandle.

**Errors:** On error, False is returned.

**See also:** CreatePipeStreams ([572](#))

## 25.4.2 CreatePipeStreams

**Synopsis:** Create a pair of pipe stream.

**Declaration:** procedure CreatePipeStreams(var InPipe: TInputPipeStream; var OutPipe: TOutputPipeStream)

**Visibility:** default

**Description:** CreatePipeStreams creates a set of pipe file descriptors with CreatePipeHandles ([571](#)), and if that call is succesfull, 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 ([571](#)) 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 ([572](#)) exception is raised.

**See also:** CreatePipeHandles ([571](#)), TInputPipeStream ([573](#)), TOutputPipeStream ([575](#))

## 25.5 EPipeCreation

### 25.5.1 Description

Exception raised when an error occurred during the creation of a pipe pair.

## 25.6 EPipeError

### 25.6.1 Description

Exception raised when an invalid operation is performed on a pipe stream.

## 25.7 EPipeSeek

### 25.7.1 Description

Exception raised when an invalid seek operation is attempted on a pipe.

## 25.8 TInputPipeStream

### 25.8.1 Description

TInputPipeStream is created by the CreatePipesStreams ([572](#)) 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 (??), CreatePipesStreams ([572](#)), TOutputPipeStream ([575](#))

### 25.8.2 Method overview

Page	Property	Description
<a href="#">573</a>	Destroy	Destroy this instance of the input pipe stream
<a href="#">574</a>	Read	Read data from the stream to a buffer.
<a href="#">574</a>	Seek	Set the current position of the stream
<a href="#">573</a>	Write	Write data to the stream.

### 25.8.3 Property overview

Page	Property	Access	Description
<a href="#">574</a>	NumBytesAvailable	r	Number of bytes available for reading.

### 25.8.4 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.

Errors: None

See also: TInputPipeStream.Create ([573](#))

### 25.8.5 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 ([574](#)), Seek ([574](#))

## 25.8.6 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 ([572](#)) exception is raised if the stream does not allow the requested seek operation.

**See also:** EPipeSeek ([572](#)), Seek ([??](#))

## 25.8.7 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 ([573](#)), Seek ([574](#))

## 25.8.8 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 ([574](#))

## 25.9 TOutputPipeStream

### 25.9.1 Description

TOutputPipeStream is created by the CreatePipeStreams ([572](#)) call to represent the writing end of a pipe. It is a TStream (??) descendent which does not allow reading.

See also: TStream (??), CreatePipeStreams ([572](#)), TInputPipeStream ([573](#))

### 25.9.2 Method overview

Page	Property	Description
<a href="#">575</a>	Destroy	Destroy this instance of the output pipe stream
<a href="#">575</a>	Read	Read data from the stream.
<a href="#">575</a>	Seek	Sets the position in the stream

### 25.9.3 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.

**Errors:** None

See also: TOutputPipeStream.Create ([575](#))

### 25.9.4 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 ([572](#)) exception is raised if this method is called.

### 25.9.5 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 ([575](#))

# Chapter 26

## Reference for unit 'pooledmm'

### 26.1 Used units

Table 26.1: Used units by unit 'pooledmm'

Name	Page
Classes	??
System	??

### 26.2 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.

### 26.3 Constants, types and variables

#### 26.3.1 Types

```
PPooledMemManagerItem = ^TPooledMemManagerItem
```

PPooledMemManagerItem is a pointer type, pointing to a TPooledMemManagerItem ([577](#)) item, used in a linked list.

```
TEnumItemsMethod = procedure(Item: Pointer) of object
```

TEnumItemsMethod is a prototype for the callback used in the TNonFreePooledMemManager.EnumerateItems ([578](#)) call. The parameter Item will be set to each of the pointers in the item list of TNonFreePooledMemManager ([577](#)).

```
TPooledMemManagerItem = record
  Next : PPooledMemManagerItem;
end
```

`TPooledMemManagerItem` is used internally by the `TPooledMemManager` (579) class to maintain the free list block. It simply points to the next free block.

## 26.4 TNonFreePooledMemManager

### 26.4.1 Description

`TNonFreePooledMemManager` keeps a list of fixed-size memory blocks in memory. Each block has the same size, making it suitable for storing a lot of records of the same type. It does not free the items stored in it, except when the list is cleared as a whole.

It allocates memory for the blocks in a exponential way, i.e. each time a new block of memory must be allocated, it's size is the double of the last block. The first block will contain 8 items.

### 26.4.2 Method overview

Page	Property	Description
577	Clear	Clears the memory
577	Create	Creates a new instance of <code>TNonFreePooledMemManager</code>
578	Destroy	Removes the <code>TNonFreePooledMemManager</code> instance from memory
578	EnumerateItems	Enumerate all items in the list
578	NewItem	Return a pointer to a new memory block

### 26.4.3 Property overview

Page	Property	Access	Description
578	ItemSize	r	Size of an item in the list

### 26.4.4 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` (578) is valid after a call to `Clear`

**See also:** `NewItem` (578)

### 26.4.5 TNonFreePooledMemManager.Create

**Synopsis:** Creates a new instance of `TNonFreePooledMemManager`

**Declaration:** constructor Create(TheItemSize: Integer)

**Visibility:** public

**Description:** `Create` creates a new instance of `TNonFreePooledMemManager` and sets the item size to `TheItemSize`.

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

**See also:** `TNonFreePooledMemManager.ItemSize` (578)

### 26.4.6 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 \(577\)](#), [TNonFreePooledMemManager.Clear \(577\)](#)

### 26.4.7 TNonFreePooledMemManager.NewItem

**Synopsis:** Return a pointer to a new memory block

**Declaration:** `function NewItem : Pointer`

**Visibility:** public

**Description:** `NewItem` returns a pointer to an unused memory block of size `ItemSize` ([578](#)). 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 \(577\)](#)

### 26.4.8 TNonFreePooledMemManager.EnumerateItems

**Synopsis:** Enumerate all items in the list

**Declaration:** `procedure EnumerateItems(const Method: TEnumItemsMethod)`

**Visibility:** public

**Description:** `EnumerateItems` will enumerate over all items in the list, passing the items to `Method`. This can be used to execute certain operations on all items in the list. (for example, simply list them)

### 26.4.9 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 \(577\)](#)

## 26.5 TPooledMemManager

### 26.5.1 Description

`TPooledMemManager` is a class which maintains a linked list of blocks, represented by the `TPooledMemManagerItem` (577) record. It should not be used directly, but should be descended from and the descendent should implement the actual memory manager.

See also: `TPooledMemManagerItem` (577)

### 26.5.2 Method overview

Page	Property	Description
579	Clear	Clears the list
579	Create	Creates a new instance of the <code>TPooledMemManager</code> class
580	Destroy	Removes an instance of <code>TPooledMemManager</code> class from memory

### 26.5.3 Property overview

Page	Property	Access	Description
581	AllocatedCount	r	Total number of allocated items in the list
580	Count	r	Number of items in the list
581	FreeCount	r	Number of free items in the list
581	FreedCount	r	Total number of freed items in the list.
580	MaximumFreeCountRatio	rw	Maximum ratio of free items over total items
580	MinimumFreeCount	rw	Minimum count of free items in the list

### 26.5.4 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` (581)

### 26.5.5 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` (580)

### 26.5.6 TPooledMemManager.Destroy

**Synopsis:** Removes an instance of TPooledMemManager class from memory

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

**Visibility:** public

**Description:** `Destroy` calls `Clear` (579) and then calls the inherited `destroy`.

`Destroy` should never be called directly. Instead `Free` should be used, or `FreeAndNil`

**See also:** `TPooledMemManager.Create` (579)

### 26.5.7 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` (580)

### 26.5.8 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` (580)

### 26.5.9 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` (581), `TPooledMemManager.AllocatedCount` (581), `TPooledMemManager.FreedCount` (581)

### **26.5.10 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 ([580](#)), TPooledMemManager.AllocatedCount ([581](#)), TPooledMemManager.FreedCount ([581](#))

### **26.5.11 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 ([580](#)), TPooledMemManager.FreeCount ([581](#)), TPooledMemManager.FreedCount ([581](#))

### **26.5.12 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 ([580](#)), TPooledMemManager.FreeCount ([581](#)), TPooledMemManager.AllocatedCount ([581](#))

# Chapter 27

## Reference for unit 'process'

### 27.1 Used units

Table 27.1: Used units by unit 'process'

Name	Page
Classes	??
Pipes	<a href="#">571</a>
System	??
sysutils	??

### 27.2 Overview

The **Process** unit contains the code for the TProcess ([586](#)) 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.

### 27.3 Constants, types and variables

#### 27.3.1 Types

```
TProcessForkEvent = procedure
```

TProcessForkEvent is the prototype for TProcess.OnForkEvent ([595](#)). 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)
```

Table 27.2: Enumeration values for type TProcessOption

Value	Explanation
poDebugOnlyThisProcess	Do not follow processes started by this process (Win32 only)
poDebugProcess	Allow debugging of the process (Win32 only)
poDefaultErrorMode	Use default error handling.
poNewConsole	Start a new console window for the process (Win32 only)
poNewProcessGroup	Start the process in a new process group (Win32 only)
poNoConsole	Do not allow access to the console window for the process (Win32 only)
poRunSuspended	Start the process in suspended state.
poStderrToOutPut	Redirect standard error to the standard output stream.
poUsePipes	Use pipes to redirect standard input and output.
poWaitOnExit	Wait for the process to terminate before returning.

When a new process is started using `TProcess.Execute` (589), 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` (583).

```
TProcessPriority = (ppHigh, ppIdle, ppNormal, ppRealTime)
```

Table 27.3: Enumeration values for type TProcessPriority

Value	Explanation
ppHigh	The process runs at higher than normal priority.
ppIdle	The process only runs when the system is idle (i.e. has nothing else to do)
ppNormal	The process runs at normal priority.
ppRealTime	The process runs at real-time priority.

This enumerated type determines the priority of the newly started process. It translates to default platform specific constants. If finer control is needed, then platform-dependent mechanism need to be used to set the priority.

```
TShowWindowOptions = (swoNone, swoHIDE, swoMaximize, swoMinimize,
                      swoRestore, swoShow, swoShowDefault,
                      swoShowMaximized, swoShowMinimized,
                      swoShowMinNOActive, swoShowNA, swoShowNoActivate,
                      swoShowNormal)
```

Table 27.4: Enumeration values for type TShowWindowOptions

Value	Explanation
swoHIDE	The main window is hidden.
swoMaximize	The main window is maximized.
swoMinimize	The main window is minimized.
swoNone	Allow system to position the window.
swoRestore	Restore the previous position.
swoShow	Show the main window.
swoShowDefault	When showing Show the main window on
swoShowMaximized	The main window is shown maximized
swoShowMinimized	The main window is shown minimized
swoShowMinNOActive	The main window is shown minimized but not activated
swoShowNA	The main window is shown but not activated
swoShowNoActivate	The main window is shown but not activated
swoShowNormal	The main window is shown normally

This type describes what the new process' main window should look like. Most of these have only effect on Windows. They are ignored on other systems.

```
TStartupOption = (suoUseShowWindow, suoUseSize, suoUsePosition,
                   suoUseCountChars, suoUseFillAttribute)
```

Table 27.5: Enumeration values for type TStartupOption

Value	Explanation
suoUseCountChars	Use the console character width as specified in TProcess (586).
suoUseFillAttribute	Use the console fill attribute as specified in TProcess (586).
suoUsePosition	Use the window sizes as specified in TProcess (586).
suoUseShowWindow	Use the Show Window options specified in TShowWindowOption (583)
suoUseSize	Use the window sizes as specified in TProcess (586)

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 (584).

### 27.3.2 Variables

TryTerminals : Array of string

TryTerminals is used under unix to test for available terminal programs in the DetectXTerm (585) function. If XTermProgram (584) 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 (585) is called.

## 27.4 Procedures and functions

### 27.4.1 CommandToList

**Synopsis:** Convert a command-line to a list of command options

**Declaration:** procedure CommandToList(S: string; List: TStrings)

**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 ([596](#))

### 27.4.2 DetectXTerm

**Synopsis:** Detect the terminal program.

**Declaration:** function DetectXTerm : string

**Visibility:** default

**Description:** DetectXTerm checks if XTermProgram ([584](#)) is set. If so, it returns that. If XTermProgram is empty, the list specified in TryTerminals ([584](#)) is tested for existence. If none is found, then the DESKTOP\_SESSION environment variable is examined:

**kdekonsole** is used if it is found.

**gnome-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 ([584](#)), TryTerminals ([584](#)), TProcess.XTermProgram ([603](#))

### 27.4.3 RunCommand

**Synopsis:** Execute a command in the current working directory

```
Declaration: function RunCommand(const exename: string;
                                const commands: Array of string;
                                var outputstring: string) : Boolean
function RunCommand(const cmdline: string; var outputstring: string)
                  : Boolean
```

**Visibility:** default

**Description:** RunCommand runs RunCommandInDir ([586](#)) with an empty current working directory.

**See also:** RunCommandInDir ([586](#))

#### 27.4.4 RunCommandIndir

**Synopsis:** Run a command in a specific directory.

```
Declaration: function RunCommandIndir(const curdir: string; const exename: string;
                                      const commands: Array of string;
                                      var outputstring: string;
                                      var exitstatus: Integer) : Integer
function RunCommandIndir(const curdir: string; const exename: string;
                        const commands: Array of string;
                        var outputstring: string) : Boolean
function RunCommandInDir(const curdir: string; const cmdline: string;
                        var 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 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.

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 ([586](#)), RunCommand ([585](#))

### 27.5 EProcess

#### 27.5.1 Description

Exception raised when an error occurs in a TProcess routine.

**See also:** TProcess ([586](#))

### 27.6 TProcess

#### 27.6.1 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 ([596](#)) property to the full pathname of the program that should be executed, and call Execute ([589](#)). To determine whether the process is still running (i.e. has not stopped executing), the Running ([600](#)) property can be checked.

More advanced techniques can be used with the Options ([598](#)) settings.

**See also:** Create ([588](#)), Execute ([589](#)), Running ([600](#)), CommandLine ([596](#)), Options ([598](#))

### **27.6.2 Method overview**

Page	Property	Description
<a href="#">589</a>	CloseInput	Close the input stream of the process
<a href="#">590</a>	CloseOutput	Close the output stream of the process
<a href="#">590</a>	CloseStderr	Close the error stream of the process
<a href="#">588</a>	Create	Create a new instance of the TProcess class.
<a href="#">589</a>	Destroy	Destroy this instance of TProcess
<a href="#">589</a>	Execute	Execute the program with the given options
<a href="#">590</a>	Resume	Resume execution of a suspended process
<a href="#">590</a>	Suspend	Suspend a running process
<a href="#">591</a>	Terminate	Terminate a running process
<a href="#">591</a>	WaitOnExit	Wait for the program to stop executing.

### 27.6.3 Property overview

Page	Property	Access	Description
<a href="#">595</a>	Active	rw	Start or stop the process.
<a href="#">595</a>	ApplicationName	rw	Name of the application to start (deprecated)
<a href="#">596</a>	CommandLine	rw	Command-line to execute (deprecated)
<a href="#">597</a>	ConsoleTitle	rw	Title of the console window
<a href="#">598</a>	CurrentDirectory	rw	Working directory of the process.
<a href="#">598</a>	Desktop	rw	Desktop on which to start the process.
<a href="#">598</a>	Environment	rw	Environment variables for the new process
<a href="#">596</a>	Executable	rw	Executable name. Supersedes CommandLine and ApplicationName.
<a href="#">594</a>	ExitStatus	r	Exit status of the process.
<a href="#">603</a>	FillAttribute	rw	Color attributes of the characters in the console window (Windows only)
<a href="#">591</a>	Handle	r	Handle of the process
<a href="#">594</a>	InheritHandles	rw	Should the created process inherit the open handles of the current process.
<a href="#">593</a>	Input	r	Stream connected to standard input of the process.
<a href="#">595</a>	OnForkEvent	rw	Event triggered after fork occurred on Linux
<a href="#">598</a>	Options	rw	Options to be used when starting the process.
<a href="#">593</a>	Output	r	Stream connected to standard output of the process.
<a href="#">597</a>	Parameters	rw	Command-line arguments. Supersedes CommandLine.
<a href="#">595</a>	PipeBufferSize	rw	Buffer size to be used when using pipes
<a href="#">599</a>	Priority	rw	Priority at which the process is running.
<a href="#">592</a>	ProcessHandle	r	Alias for Handle ( <a href="#">591</a> )
<a href="#">592</a>	ProcessID	r	ID of the process.
<a href="#">600</a>	Running	r	Determines whether the process is still running.
<a href="#">601</a>	ShowWindow	rw	Determines how the process main window is shown (Windows only)
<a href="#">600</a>	StartupOptions	rw	Additional (Windows) startup options
<a href="#">594</a>	Stderr	r	Stream connected to standard diagnostic output of the process.
<a href="#">592</a>	ThreadHandle	r	Main process thread handle
<a href="#">593</a>	ThreadID	r	ID of the main process thread
<a href="#">601</a>	WindowColumns	rw	Number of columns in console window (Windows only)
<a href="#">601</a>	WindowHeight	rw	Height of the process main window
<a href="#">602</a>	WindowLeft	rw	X-coordinate of the initial window (Windows only)
<a href="#">591</a>	WindowRect	rw	Positions for the main program window.
<a href="#">602</a>	WindowRows	rw	Number of rows in console window (Windows only)
<a href="#">602</a>	WindowTop	rw	Y-coordinate of the initial window (Windows only)
<a href="#">603</a>	WindowWidth	rw	Height of the process main window (Windows only)
<a href="#">603</a>	XTermProgram	rw	XTerm program to use (unix only)

### 27.6.4 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.

### 27.6.5 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 ([588](#))

### 27.6.6 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 ([596](#)), applying as much as of the specified options as supported on the current platform.

If the `poWaitOnExit` option is specified in Options ([598](#)), 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` ([591](#)) call can be used to wait for it to close, or the `Running` ([600](#)) call can be used to check whether it is still running.

The `TProcess.Terminate` ([591](#)) call can be used to terminate the program if it is still running, or the `Suspend` ([590](#)) call can be used to temporarily stop the program's execution.

The `ExitStatus` ([594](#)) function can be used to check the program's exit status, after it has stopped executing.

**Errors:** On error a `EProcess` ([586](#)) exception is raised.

**See also:** `TProcess.Running` ([600](#)), `TProcess.WaitOnExit` ([591](#)), `TProcess.Terminate` ([591](#)), `TProcess.Suspend` ([590](#)), `TProcess.Resume` ([590](#)), `TProcess.ExitStatus` ([594](#))

### 27.6.7 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` ([593](#)), `StdErr` ([594](#)), `Output` ([593](#)), `CloseOutput` ([590](#)), `CloseStdErr` ([590](#))

### 27.6.8 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 (593), Input (593), StdErr (594), CloseInput (589), CloseStdErr (590)

### 27.6.9 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 (593), Input (593), StdErr (594), CloseInput (589), CloseStdErr (590)

### 27.6.10 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 its execution. It should be called in particular when the poRunSuspended flag is set in Options (598).

**Errors:** None.

**See also:** TProcess.Suspend (590), TProcess.Options (598), TProcess.Execute (589), TProcess.Terminate (591)

### 27.6.11 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 (590) call.

Suspend is fundamentally different from TProcess.Terminate (591) which actually stops the process.

**Errors:** On error, a nonzero result is returned.

**See also:** TProcess.Options (598), TProcess.Resume (590), TProcess.Terminate (591), TProcess.Execute (589)

### 27.6.12 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 ([594](#)), TProcess.Suspend ([590](#)), TProcess.Execute ([589](#)), TProcess.WaitOnExit ([591](#))

### 27.6.13 TProcess.WaitOnExit

**Synopsis:** Wait for the program to stop executing.

**Declaration:** function WaitOnExit : 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.

**Errors:** On error, False is returned. No extended error information is available, as it is highly system dependent.

**See also:** TProcess.ExitStatus ([594](#)), TProcess.Terminate ([591](#)), TProcess.Running ([600](#))

### 27.6.14 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

### 27.6.15 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 ([589](#)) has been called. It is not reset after the process stopped.

See also: TProcess.ThreadHandle ([592](#)), TProcess.ProcessID ([592](#)), TProcess.ThreadID ([593](#))

### 27.6.16 TProcess.ProcessHandle

**Synopsis:** Alias for Handle ([591](#))

**Declaration:** Property ProcessHandle : THandle

**Visibility:** public

**Access:** Read

**Description:** ProcessHandle equals Handle ([591](#)) and is provided for completeness only.

See also: TProcess.Handle ([591](#)), TProcess.ThreadHandle ([592](#)), TProcess.ProcessID ([592](#)), TProcess.ThreadID ([593](#))

### 27.6.17 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 ([589](#)) has been called. It is not reset after the process stopped.

See also: TProcess.Handle ([591](#)), TProcess.ProcessID ([592](#)), TProcess.ThreadID ([593](#))

### 27.6.18 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 ([589](#)) has been called. It is not reset after the process stopped.

See also: TProcess.Handle ([591](#)), TProcess.ThreadHandle ([592](#)), TProcess.ThreadID ([593](#))

### 27.6.19 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 ([589](#)) has been called. It is not reset after the process stopped.

**See also:** TProcess.ProcessID ([592](#)), TProcess.Handle ([591](#)), TProcess.ThreadHandle ([592](#))

### 27.6.20 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 ([598](#)).

Note that writing to the stream may cause the calling process to be suspended when the created process is not reading from its input, or to cause errors when the process has terminated.

**See also:** TProcess.OutPut ([593](#)), TProcess.StdErr ([594](#)), TProcess.Options ([598](#)), TProcessOption ([583](#))

### 27.6.21 TProcess.Output

**Synopsis:** Stream connected to standard output of the process.

**Declaration:** Property Output : TInputPipeStream

**Visibility:** public

**Access:** Read

**Description:** Output is a stream which is connected to the process' standard output file handle. Anything written to standard output by the created process can be read from this stream.

The Output stream is only instantiated when the poUsePipes flag is used in Options ([598](#)).

The Output stream also contains any data written to standard diagnostic output (stderr) when the poStdErrToOutPut flag is used in Options ([598](#)).

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 ([593](#)), TProcess.StdErr ([594](#)), TProcess.Options ([598](#)), TProcessOption ([583](#))

### 27.6.22 TProcess.Stderr

**Synopsis:** Stream connected to standard diagnostic output of the process.

**Declaration:** Property Stderr : TInputPipeStream

**Visibility:** public

**Access:** Read

**Description:** StdErr is a stream which is connected to the process' standard diagnostic output file handle (StdErr). Anything written to standard diagnostic output by the created process can be read from this stream.

The StdErr stream is only instantiated when the poUsePipes flag is used in Options ([598](#)).

The Output stream equals the Output ([593](#)) when the poStdErrToOutput flag is used in Options ([598](#)).

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 ([593](#)), TProcess.Output ([593](#)), TProcess.Options ([598](#)), TProcessOption ([583](#))

### 27.6.23 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 process when it stopped executing. The value of this property is only meaningful when the process is no longer running. If it is not running then the value is zero.

**See also:** TProcess.Running ([600](#)), TProcess.Terminate ([591](#))

### 27.6.24 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 ([593](#)), TProcess.Output ([593](#)), TProcess.StdErr ([594](#))

### 27.6.25 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 (593), Input (593), StdErr (594), CloseInput (589), CloseStdErr (590), TProcessForkEvent (582)

### 27.6.26 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 (571)

### 27.6.27 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 (589), TProcess.Terminate (591)

### 27.6.28 TProcess.ApplicationName

**Synopsis:** Name of the application to start (deprecated)

**Declaration:** Property ApplicationName : string; deprecated;

**Visibility:** published

**Access:** Read,Write

**Description:** ApplicationName is an alias for TProcess.CommandLine (596). 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 (596) instead, and leave ApplicationName empty.

See also: TProcess.CommandLine (596), TProcess.Executable (596), TProcess.Parameters (597)

### 27.6.29 TProcess.CommandLine

**Synopsis:** Command-line to execute (deprecated)

**Declaration:** Property CommandLine : string; 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 (596) and TProcess.Parameters (597), 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 (595), TProcess.Executable (596), TProcess.Parameters (597)

### 27.6.30 TProcess.Executable

**Synopsis:** Executable name. Supersedes CommandLine and ApplicationName.

**Declaration:** Property Executable : string

**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 (597).

Executable supersedes the TProcess.CommandLine (596) and TProcess.ApplicationName (595) properties, which have been deprecated. However, if either of CommandLine or ApplicationName is specified, they will be used instead of Executable.

See also: CommandLine (596), ApplicationName (595), Parameters (597)

### 27.6.31 TProcess.Parameters

**Synopsis:** Command-line arguments. Supersedes CommandLine.

**Declaration:** Property Parameters : TStringList

**Visibility:** published

**Access:** Read,Write

**Description:** Parameters contains the command-line arguments that should be passed to the program specified in Executable ([596](#)).

Commandline arguments should be specified one per item in Parameters: each item in Parameters will be passed as a separate command-line item. It is 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 ([596](#)), CommandLine ([596](#)), ApplicationName ([595](#))

### 27.6.32 TProcess.ConsoleTitle

**Synopsis:** Title of the console window

**Declaration:** Property ConsoleTitle : string

**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 ([601](#)), TProcess.WindowRows ([602](#))

### 27.6.33 TProcess.CurrentDirectory

**Synopsis:** Working directory of the process.

**Declaration:** Property CurrentDirectory : string

**Visibility:** published

**Access:** Read,Write

**Description:** CurrentDirectory specifies the working directory of the newly started process.

Changing this property after the process was started has no effect.

**See also:** TProcess.Environment ([598](#))

### 27.6.34 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 ([593](#)), TProcess.Output ([593](#)), TProcess.StdErr ([594](#))

### 27.6.35 TProcess.Environment

**Synopsis:** Environment variables for the new process

**Declaration:** Property Environment : TStrings

**Visibility:** published

**Access:** Read,Write

**Description:** Environment contains the environment for the new process; it's a list of Name=Value pairs, one per line.

If it is empty, the environment of the current process is passed on to the new process.

**See also:** TProcess.Options ([598](#))

### 27.6.36 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 ([589](#)) call is made.

Table 27.6:

Option	Meaning
poRunSuspended	Start the process in suspended state.
poWaitOnExit	Wait for the process to terminate before returning.
poUsePipes	Use pipes to redirect standard input and output.
poStderrToOutput	Redirect standard error to the standard output stream.
poNoConsole	Do not allow access to the console window for the process (Win32 only)
poNewConsole	Start a new console window for the process (Win32 only)
poDefaultErrorMode	Use default error handling.
poNewProcessGroup	Start the process in a new process group (Win32 only)
poDebugProcess	Allow debugging of the process (Win32 only)
poDebugOnlyThisProcess	Do not follow processes started by this process (Win32 only)

See also: TProcessOption ([583](#)), TProcessOptions ([583](#)), TProcess.Priority ([599](#)), TProcess.StartUpOptions ([600](#))

### 27.6.37 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 27.7:

Priority	Meaning
ppHigh	The process runs at higher than normal priority.
ppIdle	The process only runs when the system is idle (i.e. has nothing else to do)
ppNormal	The process runs at normal priority.
ppRealTime	The process runs at real-time priority.

Note that not all priorities can be set by any user. Usually, only users with administrative rights (the root user on Unix) can set a higher process priority.

On unix, the process priority is mapped on Nice values as follows:

Table 27.8:

Priority	Nice value
ppHigh	20
ppIdle	20
ppNormal	0
ppRealTime	-20

See also: [TProcessPriority \(583\)](#)

### 27.6.38 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 27.9:

Priority	Meaning
suoUseShowWindow	Use the Show Window options specified in ShowWindow (601)
suoUseSize	Use the specified window sizes
suoUsePosition	Use the specified window sizes.
suoUseCountChars	Use the specified console character width.
suoUseFillAttribute	Use the console fill attribute specified in FillAttribute (603).

See also: [TProcess.ShowWindow \(601\)](#), [TProcess.WindowHeight \(601\)](#), [TProcess.WindowWidth \(603\)](#), [TProcess.WindowLeft \(602\)](#), [TProcess.WindowTop \(602\)](#), [TProcess.WindowColumns \(601\)](#), [TProcess.WindowRows \(602\)](#), [TProcess.FillAttribute \(603\)](#)

### 27.6.39 TProcess.Running

Synopsis: Determines wheter 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 \(591\)](#), [TProcess.Active \(595\)](#), [TProcess.ExitStatus \(594\)](#)

### 27.6.40 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 27.10:

Option	Meaning
swoNone	Allow system to position the window.
swoHIDE	The main window is hidden.
swoMaximize	The main window is maximized.
swoMinimize	The main window is minimized.
swoRestore	Restore the previous position.
swoShow	Show the main window.
swoShowDefault	When showing Show the main window on a default position
swoShowMaximized	The main window is shown maximized
swoShowMinimized	The main window is shown minimized
swoShowMinNOActive	The main window is shown minimized but not activated
swoShowNA	The main window is shown but not activated
swoShowNoActivate	The main window is shown but not activated
swoShowNormal	The main window is shown normally

### 27.6.41 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 ([600](#))

**See also:** TProcess.WindowHeight ([601](#)), TProcess.WindowWidth ([603](#)), TProcess.WindowLeft ([602](#)), TProcess.WindowTop ([602](#)), TProcess.WindowRows ([602](#)), TProcess.FillAttribute ([603](#)), TProcess.StartupOptions ([600](#))

### 27.6.42 TProcess.WindowHeight

**Synopsis:** Height of the process main window

**Declaration:** Property WindowHeight : Cardinal

**Visibility:** published

**Access:** Read,Write

**Description:** WindowHeight is the initial height (in pixels) of the process' main window. This property is only effective if suoUseSize is specified in StartupOptions ([600](#))

**See also:** TProcess.WindowWidth ([603](#)), TProcess.WindowLeft ([602](#)), TProcess.WindowTop ([602](#)), TProcess.WindowColumns ([601](#)), TProcess.WindowRows ([602](#)), TProcess.FillAttribute ([603](#)), TProcess.StartupOptions ([600](#))

#### 27.6.43 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 ([600](#))

**See also:** TProcess.WindowHeight ([601](#)), TProcess.WindowWidth ([603](#)), TProcess.WindowTop ([602](#)), TProcess.WindowColumns ([601](#)), TProcess.WindowRows ([602](#)), TProcess.FillAttribute ([603](#)), TProcess.StartupOptions ([600](#))

#### 27.6.44 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 ([600](#))

**See also:** TProcess.WindowHeight ([601](#)), TProcess.WindowWidth ([603](#)), TProcess.WindowLeft ([602](#)), TProcess.WindowTop ([602](#)), TProcess.WindowColumns ([601](#)), TProcess.FillAttribute ([603](#)), TProcess.StartupOptions ([600](#))

#### 27.6.45 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 ([600](#))

**See also:** TProcess.WindowHeight ([601](#)), TProcess.WindowWidth ([603](#)), TProcess.WindowLeft ([602](#)), TProcess.WindowColumns ([601](#)), TProcess.WindowRows ([602](#)), TProcess.FillAttribute ([603](#)), TProcess.StartupOptions ([600](#))

### 27.6.46 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 (600)

**See also:** TProcess.WindowHeight (601), TProcess.WindowLeft (602), TProcess.WindowTop (602), TProcess.WindowColumns (601), TProcess.WindowRows (602), TProcess.FillAttribute (603), TProcess.StartupOptions (600)

### 27.6.47 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 (601), TProcess.WindowWidth (603), TProcess.WindowLeft (602), TProcess.WindowTop (602), TProcess.WindowColumns (601), TProcess.WindowRows (602), TProcess.StartupOptions (600)

### 27.6.48 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 (598).

If none is specified, DetectXTerm (585) 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.

**gnome-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 (598), DetectXTerm (585)

# Chapter 28

## Reference for unit 'rttiutils'

### 28.1 Used units

Table 28.1: Used units by unit 'rttiutils'

Name	Page
Classes	??
StrUtils	??
System	??
sysutils	??
typinfo	??

### 28.2 Overview

The `rttiutils` unit is a unit providing simplified access to the RTTI information from published properties using the `TPropInfoList` (606) 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` (609) class can be used for this. The implementation is based on the `apputils` unit from `RXLib` by *AO ROSNO* and *Master-Bank*

### 28.3 Constants, types and variables

#### 28.3.1 Constants

```
sPropertyNameDelimiter : string = '_'
```

Separator used when constructing section/key names

#### 28.3.2 Types

```
TERaseSectEvent = procedure(const ASection: string) of object
```

TERaseSectEvent is used by TPropsStorage (609) 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.

```
TReadStrEvent = function(const ASection: string; const Item: string;
                        const Default: string) : string of object
```

TReadStrEvent is used by TPropsStorage (609) 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 (609) 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.

### 28.3.3 Variables

FindGlobalComponentCallBack : TFindComponentEvent

FindGlobalComponentCallBack is called by UpdateStoredList (606) 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.

## 28.4 Procedures and functions

### 28.4.1 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 (606), TPropsStorage.StoreObjectsProps (611) or TPropsStorage.LoadObjectsProps (611).

**See also:** ParseStoredItem (606), UpdateStoredList (606), TPropsStorage.StoreObjectsProps (611), TPropsStorage.LoadObjectsProps (611)

### 28.4.2 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 (605). 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 (605), UpdateStoredList (606), TPropsStorage.StoreObjectsProps (611), TPropsStorage.LoadObjectsProps (611)

### 28.4.3 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 (606) 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 (605) 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 (606), TPropsStorage.StoreObjectsProps (611), TPropsStorage.LoadObjectsProps (611), FindGlobalComponentCallBack (605)

## 28.5 TPropInfoList

### 28.5.1 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 (609)

**See also:** TPropsStorage (609)

### 28.5.2 Method overview

Page	Property	Description
607	Contains	Check whether a certain property is included
607	Create	Create a new instance of TPropInfoList
608	Delete	Delete property information from the list
607	Destroy	Remove the TPropInfoList instance from memory
608	Find	Retrieve property information based on name
608	Intersect	Intersect 2 property lists

### 28.5.3 Property overview

Page	Property	Access	Description
608	Count	r	Number of items in the list
609	Items	r	Indexed access to the property type pointers

### 28.5.4 TPropInfoList.Create

Synopsis: Create a new instance of TPropInfoList

Declaration: constructor Create (AObject: TObject; Filter: TTypeKinds)

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 (607)

### 28.5.5 TPropInfoList.Destroy

Synopsis: Remove the TPropInfoList instance from memory

Declaration: destructor Destroy; Override

Visibility: public

Description: Destroy cleans up the internal structures maintained by TPropInfoList and then calls the inherited Destroy.

See also: Create (607)

### 28.5.6 TPropInfoList.Contains

Synopsis: Check whether a certain property is included

Declaration: function Contains (P: PPropInfo) : Boolean

Visibility: public

Description: Contains checks whether P is included in the list of properties, and returns True if it does. If P cannot be found, False is returned.

See also: Find (608), Intersect (608)

### 28.5.7 TPropInfoList.Find

**Synopsis:** Retrieve property information based on name

**Declaration:** function Find(const AName: string) : PPropInfo

**Visibility:** public

**Description:** Find returns a pointer to the type information of the property AName. If no such information is available, the function returns Nil. The search is performed case insensitive.

**See also:** Intersect (608), Contains (607)

### 28.5.8 TPropInfoList.Delete

**Synopsis:** Delete property information from the list

**Declaration:** procedure Delete(Index: Integer)

**Visibility:** public

**Description:** Delete deletes the property information at position Index from the list. It's mainly of use in the Intersect (608) call.

**Errors:** No checking on the validity of Index is performed.

**See also:** Intersect (608)

### 28.5.9 TPropInfoList.Intersect

**Synopsis:** Intersect 2 property lists

**Declaration:** procedure Intersect(List: TPropInfoList)

**Visibility:** public

**Description:** Intersect reduces the list of properties to the ones also contained in List, i.e. all properties which are not also present in List are removed.

**See also:** Delete (608), Contains (607)

### 28.5.10 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 (609)

### 28.5.11 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 (608)

## 28.6 TPropsStorage

### 28.6.1 Description

TPropsStorage provides a mechanism to store properties from any class which has published properties (usually a TPersistent descendant) 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 (606)

### 28.6.2 Method overview

Page	Property	Description
610	LoadAnyProperty	Load a property value
611	LoadObjectsProps	Load a list of component properties
610	LoadProperties	Load a list of properties
609	StoreAnyProperty	Store a property value
611	StoreObjectsProps	Store a list of component properties
610	StoreProperties	Store a list of properties

### 28.6.3 Property overview

Page	Property	Access	Description
612	AObject	rw	Object to load or store properties from
613	OnEraseSection	rw	Erase a section in storage
613	OnReadString	rw	Read a string value from storage
613	OnWriteString	rw	Write a string value to storage
612	Prefix	rw	Prefix to use in storage
612	Section	rw	Section name for storage

### 28.6.4 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` (612) property of `TPropsStorage`.

Errors: If the property pointer is invalid or `AObject` is invalid, an exception will be raised.

See also: `AObject` (612), `LoadAnyProperty` (610), `LoadProperties` (610), `StoreProperties` (610)

### 28.6.5 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` (612) property of `TPropsStorage`.

Errors: If the property pointer is invalid or `AObject` is invalid, an exception will be raised.

See also: `AObject` (612), `StoreAnyProperty` (609), `LoadProperties` (610), `StoreProperties` (610)

### 28.6.6 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` (612) object.

Errors: If an invalid property name is specified, an exception will be raised.

See also: `AObject` (612), `StoreAnyProperty` (609), `LoadProperties` (610), `LoadAnyProperty` (610)

### 28.6.7 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` (612) object.

Errors: If an invalid property name is specified, an exception will be raised.

See also: `AObject` (612), `StoreAnyProperty` (609), `StoreProperties` (610), `LoadAnyProperty` (610)

### 28.6.8 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 (606) 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 (612) property.

**Errors:** If an invalid component is specified, an exception will be raised.

**See also:** UpdateStoredList (606), StoreObjectsProps (611), LoadProperties (610), LoadAnyProperty (610)

### 28.6.9 TPropsStorage.StoreObjectsProps

**Synopsis:** Store a list of component properties

**Declaration:** procedure StoreObjectsProps (AComponent : TComponent; StoredList : TStrings)

**Visibility:** public

**Description:** StoreObjectsProps stores a list of component properties, relative to AComponent: the names of the component properties to store are specified as follows:

```
ComponentName1.PropertyName
ComponentName2.Subcomponent1.PropertyName
```

The component instances will be located relative to AComponent, and must therefore be names of components owned by AComponent, followed by a valid property of these components. If the componentname is missing, the property name will be assumed to be a property of AComponent itself.

The Objects property of the stringlist should be filled with the instances of the components the property references refer to: they can be filled with the UpdateStoredList (606) 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 (612) property.

See also: UpdateStoredList (606), LoadObjectsProps (611), LoadProperties (610), LoadAnyProperty (610)

### 28.6.10 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 (611) or LoadObjectProps (611) will destroy any value that this property might have.

See also: LoadProperties (610), LoadAnyProperty (610), StoreProperties (610), StoreAnyProperty (609), StoreObjectProps (611), LoadObjectProps (611)

### 28.6.11 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 (612)

### 28.6.12 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 (612)

### 28.6.13 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 \(613\)](#), [OnEraseSection \(613\)](#), [TReadStrEvent \(605\)](#)

### 28.6.14 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 \(613\)](#), [OnEraseSection \(613\)](#), [TWriteStrEvent \(605\)](#)

### 28.6.15 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 \(613\)](#), [OnWriteString \(613\)](#), [TEraseSectEvent \(604\)](#)

# Chapter 29

## Reference for unit 'simpleipc'

### 29.1 Used units

Table 29.1: Used units by unit 'simpleipc'

Name	Page
Classes	??
System	??
sysutils	??

### 29.2 Overview

The SimpleIPC unit provides classes to implement a simple, one-way IPC mechanism using string messages. It provides a TSimpleIPCSERVER ([625](#)) component for the server, and a TSIMPLEIPCCCLIENT ([622](#)) 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 filesystem.

### 29.3 Constants, types and variables

#### 29.3.1 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.

```
SErrServerNotActive = 'Server with ID %s is not active.'
```

Error message if server is not active

### **29.3.2 Constants**

```
MsgVersion = 1
```

Current version of the messaging protocol

```
mtString = 1
```

String message type

```
mtUnknown = 0
```

Unknown message type

### **29.3.3 Types**

```
TIPCCClientCommClass = Class of TIPCCClientComm
```

TIPCCClientCommClass is used by TSimpleIPCCClient ([622](#)) to decide which kind of communication channel to set up.

```
TIPCServerCommClass = Class of TIPCServerComm
```

TIPCServerCommClass is used by TSimpleIPCSERVER ([625](#)) to decide which kind of communication channel to set up.

```
TMessageType = LongInt
```

TMessageType is provided for backward compatibility with earlier versions of the simpleipc unit.

```
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.

### 29.3.4 Variables

```
DefaultIPCCClientClass : TIPCCClientCommClass = Nil
```

`DefaultIPCCClientClass` is filled with a class pointer indicating which kind of communication protocol class should be instantiated by the `TSimpleIPCCClient` (622) 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).

```
DefaultIPCSERVERClass : TIPCSERVERCommClass = Nil
```

`DefaultIPCSERVERClass` is filled with a class pointer indicating which kind of communication protocol class should be instantiated by the `TSimpleIPCSERVER` (625) 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.

## 29.4 EIPCError

### 29.4.1 Description

`EIPCError` is the exception used by the various classes in the `SimpleIPC` unit to report errors.

## 29.5 TIPCCClientComm

### 29.5.1 Description

`TIPCCClientComm` is an abstract component which implements the client-side communication protocol. The behaviour expected of this class must be implemented in a platform-dependent descendent class.

The `TSimpleIPCCClient` (622) class does not implement the messaging protocol by itself. Instead, it creates an instance of a (platform dependent) descendent of `TIPCCClientComm` which handles the internals of the communication protocol.

The server side of the messaging protocol is handled by the `TIPCSERVERComm` (618) component. The descendent components must always be implemented in pairs.

See also: `TSimpleIPCCClient` (622), `TIPCSERVERComm` (618), `TSimpleIPCSERVER` (625)

### 29.5.2 Method overview

Page	Property	Description
617	Connect	Connect to the server
617	Create	Create a new instance of the <code>TIPCCClientComm</code>
617	Disconnect	Disconnect from the server
618	SendMessage	Send a message
618	ServerRunning	Check if the server is running.

### 29.5.3 Property overview

Page	Property	Access	Description
618	Owner	r	<code>TSimpleIPCCClient</code> instance for which communication must be handled.

#### 29.5.4 TIPCCClientComm.Create

**Synopsis:** Create a new instance of the TIPCCClientComm

**Declaration:** constructor Create(AOwner: TSIMPLEIPCClient); Virtual

**Visibility:** public

**Description:** Create instantiates a new instance of the TIPCCClientComm class, and stores the AOwner reference to the TSIMPLEIPCClient (622) instance for which it will handle communication. It can be retrieved later using the Owner (618) property.

**See also:** Owner (618), TSIMPLEIPCClient (622)

#### 29.5.5 TIPCCClientComm.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 (621) and ServerInstance (624) properties of the owning TSIMPLEIPCClient (622) instance.

Connect is called by the TSIMPLEIPCClient.Connect (623) call or when the Active (621) 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 (623), Active (621), Disconnect (617)

#### 29.5.6 TIPCCClientComm.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 (623) call or when the Active (621) 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 (623), Active (621), Connect (617)

### 29.5.7 TIPCCClientComm.ServerRunning

**Synopsis:** Check if the server is running.

**Declaration:** function ServerRunning : Boolean; Virtual; Abstract

**Visibility:** public

**Description:** ServerRunning returns True if the server endpoint of the communication channel can be found, or False if not. The server endpoint should be obtained from the ServerID and InstanceID properties of the owning TSsimpleIPCCClient (622) component.

**See also:** TSsimpleIPCCClient.InstanceID (622), TSsimpleIPCCClient.ServerID (622)

### 29.5.8 TIPCCClientComm.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.

### 29.5.9 TIPCCClientComm.Owner

**Synopsis:** TSsimpleIPCCClient instance for which communication must be handled.

**Declaration:** Property Owner : TSsimpleIPCCClient

**Visibility:** public

**Access:** Read

**Description:** Owner is the TSsimpleIPCCClient (622) instance for which the communication must be handled. It cannot be changed, and must be specified when the TIPCCClientComm instance is created.

**See also:** TSsimpleIPCCClient (622), TIPCCClientComm.Create (617)

## 29.6 TIPCServerComm

### 29.6.1 Description

TIPCServerComm is an abstract component which implements the server-side communication protocol. The behaviour expected of this class must be implemented in a platform-dependent descendent class.

The TSsimpleIPCServer (625) class does not implement the messaging protocol by itself. Instead, it creates an instance of a (platform dependent) descendent of TIPCServerComm which handles the internals of the communication protocol.

The client side of the messaging protocol is handled by the TIPCCClientComm (616) component. The descendent components must always be implemented in pairs.

**See also:** TSsimpleIPCServer (625), TIPCCClientComm (616)

### 29.6.2 Method overview

Page	Property	Description
619	Create	Create a new instance of the communication handler
620	PeekMessage	See if a message is available.
620	ReadMessage	Read message from the channel.
619	StartServer	Start the server-side of the communication channel
619	StopServer	Stop the server side of the communication channel.

### 29.6.3 Property overview

Page	Property	Access	Description
621	InstanceID	r	Unique identifier for the communication channel.
620	Owner	r	TSimpleIPCSERVER instance for which to handle transport

### 29.6.4 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 (620) property.

See also: Owner (620)

### 29.6.5 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 (621) property of the TSIMPLEIPCSERVER (625) instance is set to True.

Errors: In case of an error, an EIPCError (616) exception is raised.

See also: TSIMPLEIPCSERVER (625), TSIMPLEIPC.Active (621)

### 29.6.6 TIPCSERVERComm.StopServer

Synopsis: Stop the server side of the communication channel.

Declaration: procedure StopServer; Virtual; Abstract

Visibility: public

**Description:** StartServer 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 (621) property of the TSIMPLEIPCSERVER (625) instance is set to False.

**Errors:** In case of an error, an EIPCError (616) exception is raised.

**See also:** TSIMPLEIPCSERVER (625), TSIMPLEIPC.Active (621)

### 29.6.7 TIPCSERVERCOMM.PEEKMESSAGE

**Synopsis:** See if a message is available.

**Declaration:** function PeekMessage(TimeOut: Integer) : Boolean; Virtual; Abstract

**Visibility:** public

**Description:** PeekMessage can be used to see if a message is available: it returns True if a message is available. It will wait maximum TimeOut milliseconds for a message to arrive. If no message was available after this time, it will return False.

If a message was available, it can be read with the ReadMessage (620) call.

**See also:** ReadMessage (620)

### 29.6.8 TIPCSERVERCOMM.READMESSAGE

**Synopsis:** Read message from the channel.

**Declaration:** procedure ReadMessage; Virtual; Abstract

**Visibility:** public

**Description:** ReadMessage reads the message for the channel, and stores the information in the data structures in the Owner class.

ReadMessage is a blocking call: if no message is available, the program will wait till a message arrives. Use PeekMessage (620) to see if a message is available.

**See also:** TSIMPLEIPCSERVER (625)

### 29.6.9 TIPCSERVERCOMM.OWNER

**Synopsis:** TSIMPLEIPCSERVER instance for which to handle transport

**Declaration:** Property Owner : TSIMPLEIPCSERVER

**Visibility:** public

**Access:** Read

**Description:** Owner refers to the TSIMPLEIPCSERVER (625) instance for which this instance of TSIMPLEIPCSERVER handles the transport. It is specified when the TIPCSERVERCOMM is created.

**See also:** TSIMPLEIPCSERVER (625)

### 29.6.10 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.

## 29.7 TSIMPLEIPC

### 29.7.1 Description

TSimpleIPC is the common ancestor for the TSIMPLEIPCSERVER (625) and TSIMPLEIPCCCLIENT (622) classes. It implements some common properties between client and server.

See also: TSIMPLEIPCSERVER (625), TSIMPLEIPCCCLIENT (622)

### 29.7.2 Property overview

Page	Property	Access	Description
<a href="#">621</a>	Active	rw	Communication channel active
<a href="#">621</a>	ServerID	rw	Unique server identification

### 29.7.3 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 ([621](#)) identification.

See also: ServerID ([621](#))

### 29.7.4 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 ([621](#))

## 29.8 TSimpleIPCCClient

### 29.8.1 Description

TSimpleIPCCClient is the client side of the simple IPC communication protocol. The client program should create a TSimpleIPCCClient instance, set its ServerID (621) property to the unique name for the server it wants to send messages to, and then set the Active (621) property to True (or call Connect (622)).

After the connection with the server was established, messages can be sent to the server with the SendMessage (624) or SendStringMessage (624) calls.

See also: TSimpleIPCServer (625), TSimpleIPC (621), TIPCCClientComm (616)

### 29.8.2 Method overview

Page	Property	Description
<a href="#">623</a>	Connect	Connect to the server
<a href="#">622</a>	Create	Create a new instance of TSimpleIPCCClient
<a href="#">622</a>	Destroy	Remove the TSimpleIPCCClient instance from memory
<a href="#">623</a>	Disconnect	Disconnect from the server
<a href="#">624</a>	SendMessage	Send a message to the server
<a href="#">624</a>	SendStringMessage	Send a string message to the server
<a href="#">624</a>	SendStringMessageFmt	Send a formatted string message
<a href="#">623</a>	ServerRunning	Check if the server is running.

### 29.8.3 Property overview

Page	Property	Access	Description
<a href="#">624</a>	ServerInstance	rw	Server instance identification

### 29.8.4 TSimpleIPCCClient.Create

Synopsis: Create a new instance of TSimpleIPCCClient

Declaration: constructor Create(AOwner: TComponent); Override

Visibility: public

Description: Create instantiates a new instance of the TSimpleIPCCClient class. It initializes the data structures needed to handle the client side of the communication.

See also: Destroy (622)

### 29.8.5 TSimpleIPCCClient.Destroy

Synopsis: Remove the TSimpleIPCCClient 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 TSimpleIPCCClient 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 \(622\)](#)

### **29.8.6 TSimpleIPCCClient.Connect**

**Synopsis:** Connect to the server

**Declaration:** `procedure Connect`

**Visibility:** public

**Description:** `Connect` connects to the server indicated in the `ServerID` (621) and `InstanceID` (622) properties. `Connect` is called automatically if the `Active` (621) property is set to `True`.

After a successful call to `Connect`, messages can be sent to the server using `SendMessage` (624) or `SendStringMessage` (624).

Calling `Connect` if the connection is already open has no effect.

**Errors:** If creating the connection fails, an `EIPCError` (616) exception may be raised.

See also: [ServerID \(621\)](#), [InstanceID \(622\)](#), [Active \(621\)](#), [SendMessage \(624\)](#), [SendStringMessage \(624\)](#), [Disconnect \(623\)](#)

### **29.8.7 TSsimpleIPCCClient.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` (623). `Disconnect` is called automatically if the `Active` (621) 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` (616) exception may be raised.

See also: [Active \(621\)](#), [Connect \(623\)](#)

### **29.8.8 TSsimpleIPCCClient.ServerRunning**

**Synopsis:** Check if the server is running.

**Declaration:** `function ServerRunning : Boolean`

**Visibility:** public

**Description:** `ServerRunning` verifies if the server indicated in the `ServerID` (621) and `InstanceID` (622) 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 \(623\)](#)

### 29.8.9 **TSimpleIPCCClient.SendMessage**

**Synopsis:** Send a message to the server

**Declaration:** procedure SendMessage (MsgType: TMessagetype; Stream: TStream)

**Visibility:** public

**Description:** SendMessage sends a message of type MsgType and data from stream to the server. The client must be connected for this call to work.

**Errors:** In case an error occurs, or there is no connection to the server, an EIPCError (616) exception is raised.

**See also:** Connect (623), SendStringMessage (624)

### 29.8.10 **TSimpleIPCCClient.SendStringMessage**

**Synopsis:** Send a string message to the server

**Declaration:** procedure SendStringMessage (const Msg: string)  
procedure SendStringMessage (MsgType: TMessagetype; const Msg: string)

**Visibility:** public

**Description:** SendStringMessage sends a string message with type MsgTyp and data Msg to the server.  
This is a convenience function: a small wrapper around the SendMessage (624) method

**Errors:** Same as for SendMessage.

**See also:** SendMessage (624), Connect (623), SendStringMessageFmt (624)

### 29.8.11 **TSimpleIPCCClient.SendStringMessageFmt**

**Synopsis:** Send a formatted string message

**Declaration:** procedure SendStringMessageFmt (const Msg: string; Args: Array of const)  
procedure SendStringMessageFmt (MsgType: TMessagetype; const Msg: string;  
Args: Array of const)

**Visibility:** public

**Description:** SendStringMessageFmt sends a string message with type MsgTyp and message formatted from Msg and Args to the server. This is a convenience function: a small wrapper around the SendStringMessage (624) method

**Errors:** Same as for SendMessage.

**See also:** SendMessage (624), Connect (623), SendStringMessage (624)

### 29.8.12 **TSimpleIPCCClient.ServerInstance**

**Synopsis:** Server instance identification

**Declaration:** Property ServerInstance : string

**Visibility:** public

**Access:** Read,Write

**Description:** `ServerInstance` should be used in case a particular instance of the server identified with `ServerID` should be contacted. This must be used if the server has its `Global` (628) property set to `False`, and should match the server's `InstanceID` (628) property.

See also: `ServerID` (621), `Global` (628), `InstanceID` (628)

## 29.9 TSimpleIPCSERVER

### 29.9.1 Description

`TSimpleIPCSERVER` is the server side of the simple IPC communication protocol. The server program should create a `TSimpleIPCSERVER` instance, set its `ServerID` (621) property to a unique name for the system, and then set the `Active` (621) property to `True` (or call `StartServer` (626)).

After the server was started, it can check for availability of messages with the `PeekMessage` (626) call, and read the message with `ReadMessage` (625).

See also: `TSimpleIPCCClient` (622), `TSimpleIPC` (621), `TIPCSERVERComm` (618)

### 29.9.2 Method overview

Page	Property	Description
625	Create	Create a new instance of <code>TSimpleIPCSERVER</code>
626	Destroy	Remove the <code>TSimpleIPCSERVER</code> instance from memory
627	GetMessageData	Read the data of the last message in a stream
626	PeekMessage	Check if a client message is available.
626	StartServer	Start the server
626	StopServer	Stop the server

### 29.9.3 Property overview

Page	Property	Access	Description
628	Global	rw	Is the server reachable to all users or not
628	InstanceID	r	Instance ID
628	MsgData	r	Last message data
627	MsgType	r	Last message type
628	OnMessage	rw	Event triggered when a message arrives
627	StringMessage	r	Last message as a string.

### 29.9.4 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` (626)

### 29.9.5 **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 \(625\)](#)

### 29.9.6 **TSimpleIPCServer.StartServer**

**Synopsis:** Start the server

**Declaration:** `procedure StartServer`

**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` (618) descendent) and activates the communication channel.

After this method was called, clients can connect and send messages.

Prior to calling this method, the `ServerID` (621) property must be set.

**Errors:** If an error occurs a `EIPCError` (616) exception may be raised.

**See also:** [TIPCSERVERComm \(618\)](#), [Active \(621\)](#), [ServerID \(621\)](#), [StopServer \(626\)](#)

### 29.9.7 **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` (618) descendent).

**See also:** [TIPCSERVERComm \(618\)](#), [Active \(621\)](#), [ServerID \(621\)](#), [StartServer \(626\)](#)

### 29.9.8 **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 ([625](#)).

See also: ReadMessage ([625](#))

### 29.9.9 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 TSsimpleIPCServer.MsgData ([628](#)) 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 succesful call to ReadMessage ([625](#)). It will also not clear the data buffer.

See also: StringMessage ([627](#)), MsgData ([628](#)), MsgType ([627](#))

### 29.9.10 TSsimpleIPCServer.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 succesful call to ReadMessage ([625](#)).

See also: GetMessageData ([627](#))

### 29.9.11 TSsimpleIPCServer.MsgType

**Synopsis:** Last message type

**Declaration:** Property MsgType : TMessagetype

**Visibility:** public

**Access:** Read

**Description:** MsgType contains the message type of the last message.

This property will contain valid data only after a succesful call to ReadMessage ([625](#)).

See also: ReadMessage ([625](#))

### 29.9.12 **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 (627) is better suited to read the data.

This property will contain valid data only after a successful call to ReadMessage (625).

**See also:** StringMessage (627), ReadMessage (625)

### 29.9.13 **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 (625) property to form the unique server endpoint which a client should use.

**See also:** ServerID (625), Global (625)

### 29.9.14 **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 (625), InstanceID (628)

### 29.9.15 **TSimpleIPCServer.OnMessage**

**Synopsis:** Event triggered when a message arrives

**Declaration:** Property OnMessage : TNotifyEvent

**Visibility:** published

**Access:** Read,Write

**Description:** OnMessage is called by ReadMessage (625) when a message has been read. The actual message data can be retrieved with one of the StringMessage (627), MsgData (628) or MsgType (627) properties.

**See also:** StringMessage (627), MsgData (628), MsgType (627)

# Chapter 30

## Reference for unit 'sqlldb'

### 30.1 Used units

Table 30.1: Used units by unit 'sqlldb'

Name	Page
bufdataset	??
Classes	??
db	<a href="#">231</a>
sqlscript	??
System	??
sysutils	??

### 30.2 Overview

The SQLDB unit defines four main classes to handle data in SQL based databases.

1. TSQLConnection ([651](#)) 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 ([662](#)) is a #fcl.db.TDataset ([285](#)) 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 ([682](#)) 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 ([674](#)) 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](#) ([631](#)).

### 30.3 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 (285) 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 (651) 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 (682) 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 (662) component. This encapsulates an SQL statement. Any kind of SQL statement can be executed. The TSQLQuery component is a TDataset descendent: If the statement returns a result set, then it can be manipulated using the usual TDataset mechanisms.

The 3 components must be linked together: the connection must point to a default transaction (it is used to execute certain queries for metadata), the transaction component must point to a connection component. The TSQLQuery component must point to both a transaction and a database.

So in order to view the contents of a table, typically the procedure goes like this:

```
{$mode objfpc}{$h+}
uses sqldb, ibconnection;

Var
  C : TSQLConnection;
  T : TSQLTransaction;
  Q : TSQLQuery;

begin
  // Create a connection.
  C:=TIBConnection.Create(Nil);
  try
    // Set credentials.
    C.UserName:='MyUSER';
    C.Password:='Secret';
    C.DatabaseName:='/home/firebird/events.fb';
    // Create a transaction.
    T:=TSQLTransaction.Create(C);
    // Point to the database instance
    T.Database:=C;
    // Now we can open the database.
    C.Connected:=True;
    // Create a query to return data
    Q:=TSQLQuery.Create(C);
    // Point to database and transaction.
```

```

Q.Database:=C;
Q.Transaction:=T;
// Set the SQL select statement
Q.SQL.Text:='SELECT * FROM USERS';
// And now use the standard TDataset methods.
Q.Open;
While not Q.EOF do
begin
  Writeln(Q.FieldByName('U_NAME').AsString);
  Q.Next
end;
Q.Close;
finally
  C.Free;
end;
end.
```

The above code is quite simple. The connection type is `TIBConnection`, which is used for Firebird/Interbase databases. To connect to another database (for instance PostgreSQL), the exact same code could be used, but instead of a `TIBConnection`, a `TPQConnection` component must be used:

```

{$mode objfpc}{$h+}
uses sqldb, pqconnection;

Var
  C : TSQLConnection;
  T : TSQLTransaction;
  Q : TSQLQuery;

begin
  // Create a connection.
  C:=TPQConnection.Create(Nil);
```

The rest of the code remains identical.

The above code used an SQL SELECT statement and the Open method to fetch data from the database. Almost the same method applies when trying to execute other kinds of queries, such as DDL queries:

```

{$mode objfpc}{$h+}
uses sqldb, ibconnection;

Var
  C : TSQLConnection;
  T : TSQLTransaction;
  Q : TSQLQuery;

begin
  C:=TIBConnection.Create(Nil);
  try
    C.UserName:='MyUSER';
    C.Password:='Secret';
    C.DatabaseName:='/home/firebird/events.fb';
```

```

T:=TSQLTransaction.Create(C);
T.Database:=C;
C.Connected:=True;
Q:=TSQLQuery.Create(C);
Q.Database:=C;
Q.Transaction:=T;
// Set the SQL statement. SQL is a tstrings instance.
With Q.SQL do
begin
Add('CREATE TABLE USERS ( ');
Add(' U_NAME VARCHAR(50), ');
Add(' U_PASSWORD VARCHAR(50) ');
Add(' ) ');
end;
// And now execute the query using ExecSQL
// There is no result, so Open cannot be used.
Q.ExecSQL;
// Commit the transaction.
T.Commit;
finally
  C.Free;
end;
end.

```

As can be seen from the above example, the setup is the same as in the case of fetching data. Note that TSQLQuery (662) can only execute 1 SQL statement during ExecSQL. If many SQL statements must be executed, TSQLScript (674) must be used.

There is much more to TSQLQuery than explained here: it can use parameters (see UsingParams (635)) and it can automatically update the data that you edit in it (see UpdateSQLs (634)).

See also: TSQLConnection (651), TSQLTransaction (682), TSQLQuery (662), TSQLConnector (661), TSQLScript (674), UsingParams (635), UpdateSQLs (634)

## 30.4 Using the universal TSQLConnector type

The normal procedure when using SQLDB is to use one of the TSQLConnection (651) 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 (661) 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 (661) 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 (640) 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 descendants. This means that if connection-specific options must be used, they must be included in the Params (660) 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 (651), TSQLConnector (661)

## 30.5 Retrieving Schema Information

Schema Information (lists of available database objects) can be retrieved using some specialized calls in TSQLConnection (651):

- TSQLConnection.GetTableNames (654) retrieves a list of available tables. The system tables can be requested.
- TSQLConnection.GetProcedureNames (654) retrieves a list of available stored procedures.
- TSQLConnection.GetFieldNames (654) 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 (646) 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 (654).

**stSysTables** Retrieves the list of system Tables in database. This is used internally by TSQLConnection.GetTableNames (654) when the system tables are requested

**stProcedures** Retrieves a list of stored procedures in database. This is used internally by TSQLConnection.GetProcedureNames (654).

**stColumns** Retrieves the list of columns (fields) in a table. This is used internally by TSQLConnection.GetFieldNames (654).

**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).

## 30.6 Automatic generation of update SQL statements

SQLDB (more in particular, TSQLQuery (662)) 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` (672) 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` (356) 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` (672) 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` (669), `InsertSQL` (670) and `DeleteSQL` (670) properties. The new field values should be specified using params with the same name as the field. The old field values should be specified using the `OLD_` prefix to the field name. The following example demonstrates this:

```
INSERT INTO MYTABLE
    (MYFIELD,MYFIELD2)
VALUES
    (:MYFIELD,:MYFIELD2);

UPDATE MYTABLE SET
    MYFIELD=:MYFIELD
    MYFIELD2=:MYFIELD2
WHERE
    (MYFIELD=:OLD_MYFIELD);

DELETE FROM MYTABLE WHERE (MyField=:OLD_MYFIELD);
```

See also: [UsingParams](#) (635), [TSQLQuery](#) (662), [UpdateSQL](#) (669), [InsertSQL](#) (670), [DeleteSQL](#) (669)

## 30.7 Using parameters

SQLDB implements parametrized queries, simulating them if the native SQL client does not support parametrized queries. A parametrized query means that the SQL statement contains placeholders for actual values. The following is a typical example:

```
SELECT * FROM MyTable WHERE (id=:id)
```

The `:id` is a parameter with the name `id`. It does not contain a value yet. The value of the parameter will be specified separately. In SQLDB this happens through the `TParams` collection, where each element of the collection is a named parameter, specified in the SQL statement. The value can be specified as follows:

```
Params.ParamByname('id').AsInteger:=123;
```

This will tell SQLDB that the parameter `id` is of type integer, and has value 123.

SQLDB uses parameters for 3 purposes:

1. When executing a query multiple times, simply with different values, this helps increase the speed if the server supports parametrized queries: the query must be prepared only once.
2. Master-Detail relationships between datasets can be established based on a parametrized detail query: the value of the parameters in the detail query is automatically obtained from fields with the same names in the master dataset. As the user scrolls through the master dataset, the detail dataset is refreshed with the new values of the params.
3. Updating of data in the database happens through parametrized update/delete/insert statements: the `TSQLQuery.UpdateSQL` (669), `TSQLQuery.DeleteSQL` (670), `TSQLQuery.InsertSQL` (670) properties of `TSQLQuery` (662) must contain parametrized queries.

An additional advantage of using parameters is that they help to avoid SQL injection: by specifying a parameter type and value, SQLDB will automatically check whether the value is of the correct type, and will apply proper quoting when the native engine does not support parameters directly.

See also: `TSQLQuery.Params` (671), `UpdateSQLs` (634)

## 30.8 Constants, types and variables

### 30.8.1 Constants

```
DefaultSQLFormatSettings : TFormatSettings = (CurrencyFormat: 1; NegCurrFormat: 5; T
```

`DefaultSQLFormatSettings` contains the default settings used when formatting date/time and other special values in Update SQL statements generated by the various `TSQLConnection` (651) descendants.

```
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` (659).

```
SingleQuotes : TQuoteChars = (''', ''')
```

`SingleQuotes` is the set of delimiters used when using single quotes for string literals.

```
StatementTokens : Array[TStatementType] of string = ('(unknown)', 'select', 'insert'
```

`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.

### 30.8.2 Types

```
TCommitRollbackAction = (caNone, caCommit, caCommitRetaining, caRollback,
                         caRollbackRetaining)
```

Table 30.2: Enumeration values for type TCommitRollbackAction

Value	Explanation
caCommit	Commit transaction
caCommitRetaining	Commit transaction, retaining transaction context
caNone	Do nothing
caRollback	Rollback transaction
caRollbackRetaining	Rollback transaction, retaining transaction context

TCommitRollbackAction is currently unused in SQLDB.

```
TConnectionDefClass = Class of TConnectionDef
```

TConnectionDefClass is used in the RegisterConnection ([641](#)) call to register a new TConnectionDef ([641](#)) instance.

```
TConnInfoType = (citAll, citServerType, citServerVersion,
                  citServerVersionString, citClientName, citClientVersion)
```

Table 30.3: Enumeration values for type TConnInfoType

Value	Explanation
citAll	All connection information
citClientName	Client library name
citClientVersion	Client library version
citServerType	Server type description
citServerVersion	Server version as an integer number
citServerVersionString	Server version as a string

Connection information to be retrieved

```
TConnOption = (sqSupportParams, sqEscapeSlash, sqEscapeRepeat)
```

Table 30.4: Enumeration values for type TConnOption

Value	Explanation
sqEscapeRepeat	Escapes in string literals are done by repeating the character.
sqEscapeSlash	Escapes in string literals are done with backslash characters.
sqSupportParams	The connection type has native support for parameters.

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 = (detCustom, detPrepare, detExecute, detFetch, detCommit,
                 detRollBack)
```

Table 30.5: Enumeration values for type TDBEventType

Value	Explanation
detCommit	Transaction Commit message
detCustom	Custom event message
detExecute	SQLExecute message
detFetch	Fetch data message
detPrepare	SQL prepare message
detRollBack	Transaction rollback message

TDBEventType describes the type of a database event message as generated by TSQLConnection (651) through the TSQLConnection.OnLog (658) event.

```
TDBEventTypes = Set of TDBEventType
```

TDBEventTypes is a set of TDBEventType (638) values, which is used to filter the set of event messages that should be sent. The TSQLConnection.LogEvents (659) 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 (658) event handler and for the global GlobalDBLogHook (640) event handling hook. Sender will contain the TSQLConnection (651) 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 (638)

```
TQuoteChars = Array[0..1] of Char
```

TQuoteChars is an array of characters that describes the used delimiters for string values.

```
TRowCount = LargeInt
```

A type to contain a result row count.

```
TSchemaType = (stNoSchema, stTables, stSysTables, stProcedures, stColumns,
               stProcedureParams, stIndexes, stPackages, stSchemata)
```

Table 30.6: Enumeration values for type TSchemeType

Value	Explanation
stColumns	Columns in a table
stIndexes	Indexes for a table
stNoSchema	No schema
stPackages	Packages (for databases that support them)
stProcedureParams	Parameters for a stored procedure
stProcedures	Stored procedures in database
stSchemata	List of schemas in database(s)
stSysTables	System tables in database
stTables	User Tables in database

TSchemeType describes which schema information to retrieve in the TCustomSQLQuery.SetSchemaInfo (646) 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 (661)

```
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 (680) and TSQLQuery (662) 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.

```
TStatementType = (stUnknown, stSelect, stInsert, stUpdate, stDelete, stDDL,
                  stGetSegment, stPutSegment, stExecProcedure,
                  stStartTrans, stCommit, stRollback, stSelectForUpd)
```

Table 30.7: Enumeration values for type TStatementType

Value	Explanation
stCommit	The statement commits a transaction
stDDL	The statement is a SQL DDL (Data Definition Language) statement
stDelete	The statement is a SQL DELETE statement
stExecProcedure	The statement executes a stored procedure
stGetSegment	The statement is a SQL get segment statement
stInsert	The statement is a SQL INSERT statement
stPutSegment	The statement is a SQL put segment statement
stRollback	The statement rolls back a transaction
stSelect	The statement is a SQL SELECT statement
stSelectForUpd	The statement selects data for update
stStartTrans	The statement starts a transaction
stUnknown	Unknown (other) information
stUpdate	The statement is a SQL UPDATE statement

TStatementType describes the kind of SQL statement that was entered in the SQL property of a TSQLQuery (662) component.

### 30.8.3 Variables

GlobalDBLogHook : TDBLogNotifyEvent

GlobalDBLogHook can be set in addition to local TSQLConnection.Onlog (658) 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.

## 30.9 Procedures and functions

### 30.9.1 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 (641), TConnectionDef (641), TConnectionDef.TypeName (642)

### 30.9.2 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` (641). The names are the names as returned by `TConnectionDef.TypeName` (642)

See also: `RegisterConnection` (641), `TConnectionDef.TypeName` (642)

### 30.9.3 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` (641) descendent to register the connection type described in the `TConnectionDef` (641) descendent. The connection type is registered with the name as returned by `TConnectionDef.TypeName` (642).

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` (641).

Errors: if `Def` is `Nil`, access violations will occur.

See also: `TConnectionDef` (641), `UnRegisterConnection` (641)

### 30.9.4 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` (642) 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` (641), `RegisterConnection` (641)

## 30.10 TConnectionDef

### 30.10.1 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` (641). A descendent class should override at least the `TConnectionDef.TypeName` (642) and `TConnectionDef.ConnectionClass` (642) methods to return the specific name and connection class to use.

See also: `TConnectionDef.TypeName` (642), `TConnectionDef.ConnectionClass` (642), `RegisterConnection` (641)

### 30.10.2 Method overview

Page	Property	Description
644	ApplyParams	Apply parameters to an instance of TSQLConnection
642	ConnectionClass	Class to instantiate when this connection is requested
643	DefaultLibraryName	Default library name
642	Description	A descriptive text for this connection type
643	LoadedLibraryName	Currently loaded library.
643	LoadFunction	Return a function to call when the client library must be loaded
642	TypeName	Name of the connection type
643	UnLoadFunction	Return a function to call when the client library must be unloaded

### 30.10.3 TConnectionDef.TypeName

Synopsis: Name of the connection type

Declaration: class function TypeName; Virtual

Visibility: default

Description: TypeName is overridden by descendent classes to return the unique name for this connection type. It is what the TSQLConnector.ConnectorType (661) property should be set to select this connection type for the universal connection, and is the name that the GetConnectionDef (640) call will use when looking for a connection type. It must be overridden by descendants of TConnectionDef. This name is also returned in the list returned by GetConnectionList (640). This name can be an arbitrary name, no restrictions on the allowed characters exist.

See also: TSQLConnector.ConnectorType (661), GetConnectionDef (640), GetConnectionList (640), TConnectionDef.ConnectionClass (642)

### 30.10.4 TConnectionDef.ConnectionClass

Synopsis: Class to instantiate when this connection is requested

Declaration: class function ConnectionClass; 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 descendants of TConnectionDef. It may not be Nil.

See also: TConnectionDef.TypeName (642)

### 30.10.5 TConnectionDef.Description

Synopsis: A descriptive text for this connection type

Declaration: class function Description; 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 (642)

### 30.10.6 TConnectionDef.DefaultLibraryName

**Synopsis:** Default library name

**Declaration:** class function DefaultLibraryName; 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 \(638\)](#), [TConnectionDef \(641\)](#), [TLibraryUnLoadFunction \(638\)](#)

### 30.10.7 TConnectionDef.LoadFunction

**Synopsis:** Return a function to call when the client library must be loaded

**Declaration:** class function LoadFunction; 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 \(638\)](#), [TConnectionDef.UnLoadFunction \(643\)](#), [TConnectionDef.DefaultLibraryName \(643\)](#), [TConnectionDef.LoadedLibraryName \(643\)](#)

### 30.10.8 TConnectionDef.UnLoadFunction

**Synopsis:** Return a function to call when the client library must be unloaded

**Declaration:** class function UnLoadFunction; 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 \(638\)](#), [TConnectionDef.LoadFunction \(643\)](#), [TConnectionDef.DefaultLibraryName \(643\)](#), [TConnectionDef.LoadedLibraryName \(643\)](#)

### 30.10.9 TConnectionDef.LoadedLibraryName

**Synopsis:** Currently loaded library.

**Declaration:** class function LoadedLibraryName; Virtual

**Visibility:** default

**Description:** LoadedLibraryName must be overridden by descendants 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 \(638\)](#), [TLibraryUnLoadFunction \(638\)](#), [TConnectionDef.LoadFunction \(643\)](#), [TConnectionDef.UnLoadFunction \(643\)](#), [TConnectionDef.DefaultLibraryName \(643\)](#)

### 30.10.10 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 (651) 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 (642). Params contains the contents of the TSQLConnection.Params (660) property of the connector.

See also: TSQLConnection.Params (660)

## 30.11 TCustomSQLQuery

### 30.11.1 Description

TCustomSQLQuery encapsulates a SQL statement: it implements all the necessary #fcl.db.TDataset (285) 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 (645) method.

Do not instantiate a TCustomSQLQuery class directly, instead use the TSQLQuery (662) descendent.

See also: TSQLQuery (662)

### 30.11.2 Method overview

Page	Property	Description
646	Create	Create a new instance of TCustomSQLQuery.
646	Destroy	Destroy instance of TCustomSQLQuery
645	ExecSQL	Execute a SQL statement that does not return a result set
647	ParamByName	Return parameter by name
644	Prepare	Prepare a query for execution.
647	RowsAffected	Return the number of rows (records) affected by the last DML/DDL statement
646	SetSchemaInfo	SetSchemaInfo prepares the dataset to retrieve schema info.
645	UnPrepare	Unprepare a prepared query

### 30.11.3 Property overview

Page	Property	Access	Description
647	Prepared	r	Is the query prepared ?

### 30.11.4 TCustomSQLQuery.Prepare

**Synopsis:** Prepare a query for execution.

**Declaration:** procedure Prepare; Virtual

Visibility: public

**Description:** `Prepare` will prepare the SQL for execution. It will open the database connection if it was not yet open, and will start a transaction if none was started yet. It will then determine the statement type. Finally, it will pass the statement on to the database engine if it supports preparing of queries.

Strictly speaking, it is not necessary to call `prepare`, the component will prepare the statement whenever it is necessary. If a query will be executed repeatedly, it is good practice to prepare it once before starting to execute it. This will speed up execution, since resources must be allocated only once.

**Errors:** If the SQL server cannot prepare the statement, an exception will be raised.

**See also:** [TSQLQuery.StatementType \(664\)](#), [TCustomSQLQuery.UnPrepare \(645\)](#), [TCustomSQLQuery.ExecSQL \(645\)](#)

### 30.11.5 TCustomSQLQuery.UnPrepare

**Synopsis:** Unprepare a prepared query

**Declaration:** `procedure UnPrepare; Virtual`

Visibility: public

**Description:** `Unprepare` will unprepare a prepared query. This means that server resources for this statement are deallocated. After a query was unprepared, any `ExecSQL` or `Open` command will prepare the SQL statement again.

Several actions will unprepare the statement: Setting the `TSQLQuery.SQL (669)` 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 \(664\)](#), [TCustomSQLQuery.Prepare \(644\)](#), [TCustomSQLQuery.ExecSQL \(645\)](#)

### 30.11.6 TCustomSQLQuery.ExecSQL

**Synopsis:** Execute a SQL statement that does not return a result set

**Declaration:** `procedure ExecSQL; Virtual`

Visibility: public

**Description:** `ExecSQL` will execute the statement in `TSQLQuery.SQL (669)`, 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 (303)` method must be used.

The `SQL` property should be a single SQL command. To execute multiple SQL statements, use the `TSQLScript (674)` component instead.

If the statement is a DML statement, the number of deleted/updated/inserted rows can be determined using `TCustomSQLQuery.RowsAffected (647)`.

The `Database` and `Transaction` properties must be assigned before calling `ExecSQL`. Executing an empty SQL statement is also an error.

**Errors:** If the server reports an error, an exception will be raised.

**See also:** [TCustomSQLQuery.RowsAffected \(647\)](#), [TDataset.Open \(303\)](#)

### 30.11.7 TCustomSQLQuery.Create

**Synopsis:** Create a new instance of TCustomSQLQuery.

**Declaration:** constructor Create(AOwner: TComponent); Override

**Visibility:** public

**Description:** Create allocates a new instance on the heap and will allocate all resources for the SQL statement.  
After this it calls the inherited constructor.

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

**See also:** TCustomSQLQuery.Destroy ([646](#))

### 30.11.8 TCustomSQLQuery.Destroy

**Synopsis:** Destroy instance of TCustomSQLQuery

**Declaration:** destructor Destroy; Override

**Visibility:** public

**Description:** Destroy cleans up the instance, closing the dataset and freeing all allocated resources.

**See also:** TCustomSQLQuery.Create ([646](#))

### 30.11.9 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 ([654](#)) and TSQLConnection.GetProcedureNames ([654](#)) to get the necessary schema information from the database.

**See also:** TSQLConnection.GetTableNames ([654](#)), TSQLConnection.GetProcedureNames ([654](#)), RetrievingSchemaInformation ([634](#))

### 30.11.10 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 ([645](#)).

**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 ([645](#)), TSQLConnection ([651](#))

### 30.11.11 TCustomSQLQuery.ParamByName

**Synopsis:** Return parameter by name

**Declaration:** function ParamByName(const AParamName: string) : TParam

**Visibility:** public

**Description:** ParamByName is a shortcut for Params.ParamByName ([410](#)). 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 ([410](#)), TSQLQuery.Params ([671](#))

### 30.11.12 TCustomSQLQuery.Prepared

**Synopsis:** Is the query prepared ?

**Declaration:** Property Prepared : Boolean

**Visibility:** public

**Access:** Read

**Description:** Prepared is true if Prepare ([644](#)) was called for this query, and an UnPrepare ([645](#)) was not done after that (take care: several actions call UnPrepare implicitly). Initially, Prepared will be False. Calling Prepare if the query was already prepared has no effect.

**See also:** TCustomSQLQuery.Prepare ([644](#)), TCustomSQLQuery.UnPrepare ([645](#))

## 30.12 TCustomSQLStatement

### 30.12.1 Description

TCustomSQLStatement is a light-weight object that can be used to execute SQL statements on a database. It does not support result sets, and has none of the methods that a TDataset (630) component has. It can be used to execute SQL statements on a database that update data, execute stored procedures and DDL statements etc.

The TCustomSQLStatement is equivalent to TSQLQuery (662) in that it supports transactions (in the Transaction (657) property) and parameters (in the Params (660) property) and as such is a more versatile tool than executing queries using TSQLConnection.ExecuteDirect (653).

To use a TCustomSQLStatement is simple and similar to the use of TSQLQuery (662): set the Database (680) property to an existing connection component, and set the Transaction (682) property. After setting the SQL (681) property and filling Params (681), the SQL statement can be executed with the Execute (630) method.

TCustomSQLStatement is a parent class. Many of the properties are only made public (or published) in the TSQLStatement (680) class, which should be instantiated instead of the TCustomSQLStatement class.

See also: TSQLStatement (680), TDataset (630), TSQLQuery (662), TSQLStatement.Transaction (682), TSQLStatement.Params (681), Execute (630), TSQLStatement.Database (680), TSQLConnection.ExecuteDirect (653)

### 30.12.2 Method overview

Page	Property	Description
648	Create	Create a new instance of TCustomSQLStatement
649	Destroy	Destroy a TCustomSQLStatement instance.
649	Execute	Execute the SQL statement.
650	ParamByName	Find a parameter by name
649	Prepare	Prepare the statement for execution
650	RowsAffected	Number of rows affected by the SQL statement.
649	Unprepare	Unprepare a previously prepared statement

### 30.12.3 Property overview

Page	Property	Access	Description
650	Prepared	r	Is the statement prepared or not

### 30.12.4 TCustomSQLStatement.Create

Synopsis: Create a new instance of TCustomSQLStatement

Declaration: constructor Create (AOwner: TComponent);   Override

Visibility: public

Description: Create initializes a new instance of TCustomSQLStatement and sets the SQL (681)Params (681), ParamCheck (681) and ParseSQL (681) to their initial values.

See also: TSQLStatement.SQL (681), TSQLStatement.Params (681), TSQLStatement.ParamCheck (681), TSQLStatement.ParseSQL (681), Destroy (630)

### 30.12.5 TCustomSQLStatement.Destroy

**Synopsis:** Destroy a TCustomSQLStatement instance.

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

**Visibility:** public

**Description:** `Destroy` disconnects the TCustomSQLStatement instance from the transaction and database, and then frees the memory taken by the instance and its properties.

**See also:** [TSQLStatement.Database \(680\)](#), [TSQLStatement.Transaction \(682\)](#)

### 30.12.6 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 (630)` 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 ([681](#)) statement set or the Database ([680](#)) or Transaction ([682](#)) properties are empty.

**See also:** [TSQLStatement.SQL \(681\)](#), [TSQLStatement.Database \(680\)](#), [TSQLStatement.Transaction \(682\)](#), [Execute \(630\)](#)

### 30.12.7 TCustomSQLStatement.Execute

**Synopsis:** Execute the SQL statement.

**Declaration:** `procedure Execute`

**Visibility:** public

**Description:** `Execute` executes the SQL ([630](#)) 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 ([681](#)) statement set or the Database ([680](#)) or Transaction ([682](#)) 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 \(681\)](#), [TSQLStatement.Database \(680\)](#), [TSQLStatement.Transaction \(682\)](#)

### 30.12.8 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 (681), TSQLStatement.Database (680), TSQLStatement.Transaction (682), Prepare (630)

### 30.12.9 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 (681) property.

**Errors:** If no parameter with the given name is found, an exception is raised.

**See also:** TSQLStatement.Params (681), TParams.ParamByname (630)

### 30.12.10 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 (630) was called. Not all databases may support this.

**See also:** Execute (630)

### 30.12.11 TCustomSQLStatement.Prepared

**Synopsis:** Is the statement prepared or not

**Declaration:** Property Prepared : Boolean

**Visibility:** public

**Access:** Read

**Description:** Prepared equals True if Prepare (630) was called (implicitly or explicitly), it returns False if not. It can be set to True or False to call Prepare (630) or UnPrepare (630), respectively.

**See also:** Prepare (630), UnPrepare (630)

## 30.13 TServerIndexDefs

### 30.13.1 Description

`TServerIndexDefs` is a simple descendent of `TIndexDefs` (380) that implements the necessary methods to update the list of definitions using the `TSQLConnection` (651). It should not be used directly.

See also: `TSQLConnection` (651)

### 30.13.2 Method overview

Page	Property	Description
<a href="#">651</a>	Create	Create a new instance of <code>TServerIndexDefs</code>
<a href="#">651</a>	Update	Updates the list of indexes

### 30.13.3 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` (644) descendent.

Errors: An `EDatabaseError` exception will be raised if `ADataSet` is not a `TCustomSQLQuery` (644) descendent.

### 30.13.4 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` (651) methods for this.

## 30.14 TSQLConnection

### 30.14.1 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` (662), `TSQLTransaction` (682)

### 30.14.2 Method overview

Page	Property	Description
652	Create	Create a new instance of TSQLConnection
655	CreateDB	Create a new Database on the server
653	Destroy	Destroys the instance of the connection.
655	DropDB	Procedure to drop or remove a Database
653	EndTransaction	End the Transaction associated with this connection
653	ExecuteDirect	Execute a piece of SQL code directly, using a Transaction if specified
655	GetConnectionString	Return some information about the connection
654	GetFieldNames	Gets a list of the field names in the specified table
654	GetProcedureNames	Gets a list of Stored Procedures in the Database
655	GetSchemaNames	Get database schema names
654	GetTableNames	Get a list of the tables in the specified database
653	StartTransaction	Start the Transaction associated with this Connection

### 30.14.3 Property overview

Page	Property	Access	Description
658	CharSet	rw	The character set to be used in this database
659	Connected		Is a connection to the server active or not
656	ConnOptions	r	The set of Connection options being used in the Connection
660	DatabaseName		The name of the database to which connection is required.
656	FieldNameQuoteChars	rw	Characters used to quote field names.
656	Handle	r	Low level handle used by the connection.
658	HostName	rw	The name of the host computer where the database resides
660	KeepConnection		Attempt to keep the connection open once it is established.
659	LogEvents	rw	Filter for events to log
660	LoginPrompt		Should SQLDB prompt for user credentials when a connection is activated.
658	OnLog	rw	Event handler for logging events
661	OnLogin		Event handler for login process
660	Params		Extra connection parameters
657	Password	rw	Password used when authenticating on the database server
659	Role	rw	Role in which the user is connecting to the database
657	Transaction	rw	Default transaction to be used for this connection
657	UserName	rw	The username for authentication on the database server

### 30.14.4 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 (651). After calling the inherited constructor, it will initialize the FieldNameQuoteChars (656) property and some other fields for internal use.

See also: [FieldNameQuoteChars](#) (656)

### 30.14.5 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.

### 30.14.6 TSQLConnection.StartTransaction

Synopsis: Start the Transaction associated with this Connection

Declaration: `procedure StartTransaction;   Override`

Visibility: public

Description: `StartTransaction` is a convenience method which starts the default transaction (`TSQLConnection.Transaction` (657)). It is equivalent to

`Connection.Transaction.StartTransaction`

Errors: If no transaction is assigned, an exception will be raised.

See also: [EndTransaction](#) (653)

### 30.14.7 TSQLConnection.EndTransaction

Synopsis: End the Transaction associated with this connection

Declaration: `procedure EndTransaction;   Override`

Visibility: public

Description: `StartTransaction` is a convenience method which ends the default transaction (`TSQLConnection.Transaction` (657)). It is equivalent to

`Connection.Transaction.EndTransaction`

Errors: If no transaction is assigned, an exception will be raised.

See also: [StartTransaction](#) (653)

### 30.14.8 TSQLConnection.ExecuteDirect

Synopsis: Execute a piece of SQL code directly, using a Transaction if specified

Declaration: `procedure ExecuteDirect(SQL: string);   Virtual;   Overload`  
`procedure ExecuteDirect(SQL: string; ATransaction: TSQLTransaction)`  
                  `;   Virtual;   Overload`

Visibility: public

**Description:** ExecuteDirect executes an SQL statement directly. If ATransaction is Nil then the default transaction is used, otherwise the specified transaction is used.

ExecuteDirect does not offer support for parameters, so only statements that do not need parsing and parameters substitution can be handled. If parameter substitution is required, use a TSQLQuery (662) component and its ExecSQL (645) method.

**Errors:** If no transaction is assigned, and no transaction is passed, an exception will be raised.

See also: TSQLQuery (662), ExecSQL (645)

### 30.14.9 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 (654), TSQLConnection.GetFieldNames (654)

### 30.14.10 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 (654), TSQLConnection.GetFieldNames (654)

### 30.14.11 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 (654), TSQLConnection.GetProcedureNames (654)

### 30.14.12 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 (654), TSQLConnection.GetProcedureNames (654), TSQLConnection.GetFieldNames (654)

### 30.14.13 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 (637)

### 30.14.14 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 (660) property, the user credentials are taken from the TSQLConnection.UserName (657) and TSQLConnection.Password (657) 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 (655)

### 30.14.15 TSQLConnection.DropDB

**Synopsis:** Procedure to drop or remove a Database

**Declaration:** procedure DropDB; Virtual

**Visibility:** public

**Description:** DropDB does the opposite of CreateDB (655). 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` ([655](#))

### 30.14.16 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.

### 30.14.17 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 ([651](#)) descendent to the default for that particular connection type.

See also: `TSQLConnection` ([651](#))

### 30.14.18 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` ([637](#))

### 30.14.19 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 ([657](#)) 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 ([657](#)), TSQLConnection.HostName ([658](#))

### 30.14.20 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 ([682](#)) 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 ([654](#))

See also: TSQLTransaction ([682](#))

### 30.14.21 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 ([657](#)), TSQLConnection.HostName ([658](#)), TSQLConnection.Role ([659](#)), TSQLConnection.Charset ([658](#))

### 30.14.22 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 (657), TSQLConnection.HostName (658), TSQLConnection.UserName (657), TSQLConnection.Role (659)

### 30.14.23 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 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 (659), TSQLConnection.Password (657), TSQLConnection.UserName (657), TSQLConnection.DatabaseName (660), TSQLConnection.Charset (658)

### 30.14.24 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, committ 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 (659) 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 (640) event handler.

**See also:** GlobalDBLogHook (640), TSQLConnection.LogEvents (659)

### **30.14.25 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 ([658](#)) and GlobalDBLogHook ([640](#)) event handlers. Only event types that are listed in this property will be sent.

**See also:** GlobalDBLogHook ([640](#)), TSQLConnection.OnLog ([658](#))

### **30.14.26 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 ([660](#)) on host HostName ([658](#)) using the credentials specified in UserName ([657](#)) and Password ([657](#)). If the connection or authentication fails, an exception is raised. This has the same effect as calling Open ([272](#)).

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 ([630](#))

**See also:** TSQLConnection.Password ([657](#)), TSQLConnection.UserName ([657](#)), TSQLConnection.DatabaseName ([660](#)), TSQLConnection.Role ([659](#))

### **30.14.27 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 ([657](#)), TSQLConnection.UserName ([657](#)), TSQLConnection.DatabaseName ([660](#)), TSQLConnection.Hostname ([658](#))

### 30.14.28 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 ([657](#)), TSQLConnection.UserName ([657](#)), TSQLConnection.Charset ([658](#)), TSQLConnection.Hostname ([658](#))

### 30.14.29 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.

### 30.14.30 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 ([661](#)) event handler. The UserName ([657](#)) and Password ([657](#)) properties are ignored in this case.

**See also:** TSQLConnection.Password ([657](#)), TSQLConnection.UserName ([657](#)), OnLogin ([661](#))

### 30.14.31 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 ([657](#)), TSQLConnection.UserName ([657](#)), TSQLConnection.Hostname ([658](#)), TSQLConnection.DatabaseName ([660](#))

### 30.14.32 TSQLConnection.OnLogin

**Synopsis:** Event handler for login process

**Declaration:** Property OnLogin :

**Visibility:** published

**Access:**

**Description:** OnLogin will be used when loginPrompt ([660](#)) 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 ([660](#))

## 30.15 TSQLConnector

### 30.15.1 Description

TSQLConnector implements a general connection type. When switching database backends, the normal procedure is to replace one instance of TSQLConnection ([651](#)) descendent with another, and connect all instances of TSQLQuery ([662](#)) and TSQLTransaction ([682](#)) to the new connection.

Using TSQLConnector avoids this: the type of connection can be set using the ConnectorType ([661](#)) property, which is a string property. The TSQLConnector class will (in the background) create the correct TSQLConnection ([651](#)) 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 ([661](#)), UniversalConnectors ([633](#))

### 30.15.2 Property overview

Page	Property	Access	Description
<a href="#">661</a>	ConnectorType	rw	Name of the connection type to use

### 30.15.3 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 availabe connector types in the application. The list of possible connector types can be retrieved using GetConnectionList ([640](#)) 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 \(640\)](#)

## 30.16 TSQLCursor

### 30.16.1 Description

`TSQLCursor` is an abstract internal object representing a result set returned by a single SQL select statement (`TSQLHandle (662)`). statement. It is used by the `TSQLQuery (662)` component to handle result sets returned by SQL statements.

This object must not be used directly.

See also: [TSQLQuery \(662\)](#), [TSQLHandle \(662\)](#)

## 30.17 TSQLHandle

### 30.17.1 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 (662)` component to communicate with the `TSQLConnection (651)` descendent.

This object must not be used directly.

See also: [TSQLQuery \(662\)](#), [TSQLCursor \(662\)](#)

## 30.18 TSQLQuery

### 30.18.1 Description

`TSQLQuery` exposes the properties and some methods introduced in `TCustomSQLQuery (644)`. It encapsulates a single SQL statement: it implements all the necessary `#fcl.db.TDataset (285)` 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 (645)` method.

Typically, the `TSQLQuery.Database (668)` property must be set once, the `TSQLQuery.Transaction (669)` property as well. Then the `TSQLQuery.SQL (669)` property can be set. Depending on the kind of SQL statement, the `Open (303)` 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 \(682\)](#), [TSQLConnection \(651\)](#), [TCustomSQLQuery.ExecSQL \(645\)](#), [TSQLQuery.SQL \(669\)](#)

### 30.18.2 Property overview

Page	Property	Access	Description
<a href="#">665</a>	Active		
<a href="#">665</a>	AfterCancel		
<a href="#">665</a>	AfterClose		
<a href="#">665</a>	AfterDelete		
<a href="#">666</a>	AfterEdit		
<a href="#">666</a>	AfterInsert		
<a href="#">666</a>	AfterOpen		
<a href="#">666</a>	AfterPost		
<a href="#">666</a>	AfterScroll		
<a href="#">665</a>	AutoCalcFields		
<a href="#">666</a>	BeforeCancel		
<a href="#">666</a>	BeforeClose		
<a href="#">667</a>	BeforeDelete		
<a href="#">667</a>	BeforeEdit		
<a href="#">667</a>	BeforeInsert		
<a href="#">667</a>	BeforeOpen		
<a href="#">667</a>	BeforePost		
<a href="#">667</a>	BeforeScroll		
<a href="#">668</a>	Database		The TSQLConnection instance on which to execute SQL Statements
<a href="#">673</a>	DataSource		Source for parameter values for unbound parameters
<a href="#">670</a>	DeleteSQL		Statement to be used when inserting a new row in the database
<a href="#">664</a>	FieldDefs		List of field definitions.
<a href="#">665</a>	Filter		
<a href="#">665</a>	Filtered		
<a href="#">671</a>	IndexDefs		List of local index Definitions
<a href="#">670</a>	InsertSQL		Statement to be used when inserting a new row in the database
<a href="#">664</a>	MaxIndexesCount		Maximum allowed number of indexes.
<a href="#">667</a>	OnCalcFields		
<a href="#">668</a>	OnDeleteError		
<a href="#">668</a>	OnEditError		
<a href="#">668</a>	OnFilterRecord		
<a href="#">668</a>	OnNewRecord		
<a href="#">668</a>	OnPostError		
<a href="#">671</a>	ParamCheck		Should the SQL statement be checked for parameters
<a href="#">671</a>	Params		Parameters detected in the SQL statement.
<a href="#">672</a>	ParseSQL		Should the SQL statement be parsed or not
<a href="#">669</a>	ReadOnly		
<a href="#">664</a>	SchemaType		Schema type
<a href="#">673</a>	ServerFilter		Append server-side filter to SQL statement
<a href="#">673</a>	ServerFiltered		Should server-side filter be applied
<a href="#">674</a>	ServerIndexDefs		List of indexes on the primary table of the query
<a href="#">669</a>	SQL		The SQL statement to execute
<a href="#">664</a>	StatementType		SQL statement type
<a href="#">669</a>	Transaction		Transaction in which to execute SQL statements
<a href="#">672</a>	UpdateMode		How to create update SQL statements.
<a href="#">669</a>	UpdateSQL		Statement to be used when updating an existing row in the database
<a href="#">672</a>	UsePrimaryKeyAsKey		Should primary key fields be marked <code>pfInKey</code>

### 30.18.3 TSQLQuery.SchemaType

Synopsis: Schema type

Declaration: Property SchemaType :

Visibility: public

Access:

Description: SchemaType is the schema type set by TCustomSQLQuery.SetSchemaInfo (646). It determines what kind of schema information will be returned by the TSQLQuery instance.

See also: TCustomSQLQuery.SetSchemaInfo (646), RetrievingSchemaInformation (634)

### 30.18.4 TSQLQuery.StatementType

Synopsis: SQL statement type

Declaration: Property StatementType :

Visibility: public

Access:

Description: StatementType is determined during the Prepare (644) call when ParseSQL (672) is set to True. It gives an indication of the type of SQL statement that is being executed.

See also: TSQLQuery.SQL (669), TSQLQuery.ParseSQL (672), TSQLQuery.Params (671)

### 30.18.5 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.

### 30.18.6 TSQLQuery.FieldDefs

Synopsis: List of field definitions.

Declaration: Property FieldDefs :

Visibility: published

Access:

### **30.18.7 TSQLQuery.Active**

Declaration: Property Active :

Visibility: published

Access:

### **30.18.8 TSQLQuery.AutoCalcFields**

Declaration: Property AutoCalcFields :

Visibility: published

Access:

### **30.18.9 TSQLQuery.Filter**

Declaration: Property Filter :

Visibility: published

Access:

### **30.18.10 TSQLQuery.Filtered**

Declaration: Property Filtered :

Visibility: published

Access:

### **30.18.11 TSQLQuery.AfterCancel**

Declaration: Property AfterCancel :

Visibility: published

Access:

### **30.18.12 TSQLQuery.AfterClose**

Declaration: Property AfterClose :

Visibility: published

Access:

### **30.18.13 TSQLQuery.AfterDelete**

Declaration: Property AfterDelete :

Visibility: published

Access:

### **30.18.14 TSQLQuery.AfterEdit**

Declaration: Property AfterEdit :

Visibility: published

Access:

### **30.18.15 TSQLQuery.AfterInsert**

Declaration: Property AfterInsert :

Visibility: published

Access:

### **30.18.16 TSQLQuery.AfterOpen**

Declaration: Property AfterOpen :

Visibility: published

Access:

### **30.18.17 TSQLQuery.AfterPost**

Declaration: Property AfterPost :

Visibility: published

Access:

### **30.18.18 TSQLQuery.AfterScroll**

Declaration: Property AfterScroll :

Visibility: published

Access:

### **30.18.19 TSQLQuery.BeforeCancel**

Declaration: Property BeforeCancel :

Visibility: published

Access:

### **30.18.20 TSQLQuery.BeforeClose**

Declaration: Property BeforeClose :

Visibility: published

Access:

### **30.18.21 TSQLQuery.BeforeDelete**

Declaration: Property BeforeDelete :

Visibility: published

Access:

### **30.18.22 TSQLQuery.BeforeEdit**

Declaration: Property BeforeEdit :

Visibility: published

Access:

### **30.18.23 TSQLQuery.BeforeInsert**

Declaration: Property BeforeInsert :

Visibility: published

Access:

### **30.18.24 TSQLQuery.BeforeOpen**

Declaration: Property BeforeOpen :

Visibility: published

Access:

### **30.18.25 TSQLQuery.BeforePost**

Declaration: Property BeforePost :

Visibility: published

Access:

### **30.18.26 TSQLQuery.BeforeScroll**

Declaration: Property BeforeScroll :

Visibility: published

Access:

### **30.18.27 TSQLQuery.OnCalcFields**

Declaration: Property OnCalcFields :

Visibility: published

Access:

### 30.18.28 TSQLQuery.OnDeleteError

Declaration: Property OnDeleteError :

Visibility: published

Access:

### 30.18.29 TSQLQuery.OnEditError

Declaration: Property OnEditError :

Visibility: published

Access:

### 30.18.30 TSQLQuery.OnFilterRecord

Declaration: Property OnFilterRecord :

Visibility: published

Access:

### 30.18.31 TSQLQuery.OnNewRecord

Declaration: Property OnNewRecord :

Visibility: published

Access:

### 30.18.32 TSQLQuery.OnPostError

Declaration: Property OnPostError :

Visibility: published

Access:

### 30.18.33 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 (651)) 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 (669) property must be set.

Multiple TSQLQuery instances can be connected to a database at the same time.

See also: TSQLQuery.Transaction (669), TSQLConnection (651), TSQLTransaction (682)

### 30.18.34 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 (682)) 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 (668), TSQLConnection (651), TSQLTransaction (682)

### 30.18.35 TSQLQuery.ReadOnly

**Declaration:** Property ReadOnly :

Visibility: published

Access:

### 30.18.36 TSQLQuery.SQL

**Synopsis:** The SQL statement to execute

**Declaration:** Property SQL :

Visibility: published

Access:

**Description:** SQL is the SQL statement that will be executed when ExecSQL (645) is called, or Open (303) is called. It should contain a valid SQL statement for the connection to which the TSQLQuery (662) 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 (645)

If ParseSQL (672) is True, the SQL statement will be parsed and the Params (671) property will be updated with the names of the parameters found in the SQL statement.

See also Using parameters

See also: TSQLQuery.ParseSQL (672), TSQLQuery.Params (671), TCustomSQLQuery.ExecSQL (645), TDataset.Open (303)

### 30.18.37 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 parametrized according to the conventions for specifying parameters.  
Note that old field values can be specified as :`OLD_FIELDNAME`

See also: `TSQLQuery.SQL` (669), `TSQLQuery.InsertSQL` (670), `TSQLQuery.DeleteSQL` (670), `TSQLQuery.UpdateMode` (672), `UsingParams` (635), `UpdateSQLS` (634)

### 30.18.38 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 parametrized according to the conventions for specifying parameters.  
Note that old field values can be specified as :`OLD_FIELDNAME`

See also: `TSQLQuery.SQL` (669), `TSQLQuery.UpdateSQL` (669), `TSQLQuery.DeleteSQL` (670), `TSQLQuery.UpdateMode` (672), `UsingParams` (635), `UpdateSQLS` (634)

### 30.18.39 TSQLQuery.DeleteSQL

Synopsis: Statement to be used when inserting 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 insert can be specified here.

The SQL statement should be parametrized according to the conventions for specifying parameters.  
Note that old field values can be specified as :`OLD_FIELDNAME`

See also: `TSQLQuery.SQL` (669), `TSQLQuery.UpdateSQL` (669), `TSQLQuery.DeleteSQL` (670), `TSQLQuery.UpdateMode` (672), `UsingParams` (635), `UpdateSQLS` (634)

### 30.18.40 TSQLQuery.IndexDefs

**Synopsis:** List of local index Definitions

**Declaration:** Property IndexDefs :

Visibility: published

Access:

Description: List of local index Definitions

See also: TCustomBufDataset.IndexDefs ([630](#))

### 30.18.41 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 ([672](#)) is True. For each named parameter in the SQL ([669](#)) property, a named item will appear in the collection, and the collection will be used to retrieve values from.

When Open ([303](#)) or ExecSQL ([645](#)) is called, and the Datasource ([673](#)) property is not `Nil`, then for each parameter for which no value was explicitly set (its Bound ([403](#)) 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 ([669](#)), TSQLQuery.ParseSQL ([672](#)), TParam.Bound ([403](#)), UsingParams ([635](#)), UpdateSQLS ([634](#))

### 30.18.42 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 ([395](#)) instances should be added to the Params ([630](#)) 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 ([395](#)), Params ([630](#)), TSQLStatement.ParamCheck ([681](#))

### 30.18.43 TSQLQuery.ParseSQL

**Synopsis:** Should the SQL statement be parsed or not

**Declaration:** Property ParseSQL :

Visibility: published

Access:

**Description:** ParseSQL can be set to False to prevent TSQLQuery from parsing the SQL (669) property and attempting to detect the statement type or updating the Params (671) or StatementType (664) properties.

This can be used when SQLDB has problems parsing the SQL statement, or when the SQL statement contains parameters that are part of a DDL statement such as a CREATE PROCEDURE statement to create a stored procedure.

Note that in this case the statement will be passed as-is to the SQL engine, no parameter values will be passed on.

See also: TSQLQuery.SQL (669), TSQLQuery.Params (671)

### 30.18.44 TSQLQuery.UpdateMode

**Synopsis:** How to create update SQL statements.

**Declaration:** Property UpdateMode :

Visibility: published

Access:

**Description:** UpdateMode determines how the WHERE clause of the UpdateSQL (669) and DeleteSQL (670) statements are auto-generated.

**upWhereAll**Use all old field values

**upWhereChanged**Use only old field values of modified fields

**upWhereKeyOnly**Only use key fields in the where clause.

See also: TSQLQuery.UpdateSQL (669), TSQLQuery.InsertSQL (670)

### 30.18.45 TSQLQuery.UsePrimaryKeyAsKey

**Synopsis:** Should primary key fields be marked pfInKey

**Declaration:** Property UsePrimaryKeyAsKey :

Visibility: published

Access:

**Description:** UsePrimaryKeyAsKey can be set to True to let TSQLQuery fetch all server indexes and if there is a primary key, update the ProviderFlags (356) of the fields in the primary key with pfInKey (243).

The effect of this is that when UpdateMode (672) 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 (634).

See also: TSQLQuery.UpdateMode (672), TCustomBufDataset.Unidirectional (630), TField.ProviderFlags (356), pfInKey (243), UpdateSQLs (634)

### 30.18.46 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 (303) or ExecSQL (645) is called, and the Datasource property is not Nil then for each parameter for which no value was explicitly set (its Bound (403) 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 (671), ExecSQL (645), UsingParams (635), TParam.Bound (403)

### 30.18.47 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 (669), when ServerFiltered (673) is set to True. If ServerFiltered (673) is set to False, ServerFilter is ignored.

If the dataset is active and ServerFiltered (673) is set to true, then changing this property will re-fetch the data from the server.

This property cannot be used when ParseSQL (672) 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 (672) is False will result in an exception.

**See also:** ServerFiltered (673)

### 30.18.48 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 (673). 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 (672) is False will result in an exception.

**See also:** ParseSQL (672), ServerFilter (673)

### 30.18.49 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 (672) is True. If a primary key is found, then the fields in it will be marked

See also: UsePrimaryKeyAsKey (672), Prepare (644)

## 30.19 TSQLScript

### 30.19.1 Description

TSQLScript is a component that can be used to execute many SQL statements using a TSQLQuery (662) component. The SQL statements are specified in a script TSQLScript.Script (677) separated by a terminator character (typically a semicolon (;)).

See also: TSQLTransaction (682), TSQLConnection (651), TCustomSQLQuery.ExecSQL (645), TSQLQuery.SQL (669)

### 30.19.2 Method overview

Page	Property	Description
674	Create	Create a new TSQLScript instance.
675	Destroy	Remove the TSQLScript instance from memory.
675	Execute	Execute the script.
675	ExecuteScript	Convenience function, simply calls Execute

### 30.19.3 Property overview

Page	Property	Access	Description
678	CommentsinSQL		Should comments be passed to the SQL engine ?
676	DataBase	rw	Database on which to execute the script
677	Defines		Defined macros
676	Directives		List of directives
676	OnDirective	rw	Event handler if a directive is encountered
679	OnException		Exception handling event
677	Script		The script to execute
677	Terminator		Terminator character.
676	Transaction	rw	Transaction to use in the script
678	UseCommit		Control automatic handling of the COMMIT command.
679	UseDefines		Automatically handle pre-processor defines
678	UseSetTerm		Should the SET TERM directive be recognized

### 30.19.4 TSQLScript.Create

**Synopsis:** Create a new TSQLScript instance.

**Declaration:** constructor Create(AOwner: TComponent); Override

**Visibility:** public

**Description:** Create instantiates a TSQLQuery (662) instance which will be used to execute the queries, and then calls the inherited constructor.

**See also:** TSQLScript.Destroy (675)

### 30.19.5 TSQLScript.Destroy

**Synopsis:** Remove the TSQLScript instance from memory.

**Declaration:** destructor Destroy; Override

**Visibility:** public

**Description:** Destroy frees the TSQLQuery (662) instance that was created during the Create constructor from memory and then calls the inherited destructor.

**See also:** TSQLScript.Create (674)

### 30.19.6 TSQLScript.Execute

**Synopsis:** Execute the script.

**Declaration:** procedure Execute; Override

**Visibility:** public

**Description:** Execute will execute the statements specified in Script (677) 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 (679) event handler is set, it may stop the event handler.

**Errors:** Handle errors using TSQLScript.OnException (679).

**See also:** Script (677), TSQLScript.OnException (679)

### 30.19.7 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.

### 30.19.8 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 (651) descendent. All SQL statements in the Script (677) property will be executed on this database.

**See also:** TSQLConnection (651), TSQLScript.Transaction (676), TSQLScript.Script (677)

### 30.19.9 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 (683) method.

**See also:** TSQLTransaction (682), TSQLTransaction.CommitRetaining (683), TSQLScript.Database (676)

### 30.19.10 TSQLScript.OnDirective

**Synopsis:** Event handler if a directive is encountered

**Declaration:** Property OnDirective : TSQLScriptDirectiveEvent

**Visibility:** published

**Access:** Read,Write

**Description:** OnDirective is called when a directive is encountered. When parsing the script, the script engine checks the first word of the statement. If it matches one of the words in Directives (676) 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 (676)

### 30.19.11 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 ([676](#)) event handler. The list should contain one word per line, no spaces allowed.

See also: OnDirective ([676](#))

### **30.19.12 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 ([679](#)) 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 ([679](#))

### **30.19.13 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 ([677](#)) property. Each of the statement will be executed on the database specified in Database ([676](#)). using the equivalent of the TCustomSQLQuery.ExecSQL ([645](#)) 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 ([678](#)) property. If that property is False, comments will be stripped prior to executing the SQL statements.

See also: TSQLScript.CommentsinSQL ([678](#)), TSQLScript.Terminator ([677](#)), TSQLScript.DataBase ([676](#))

### **30.19.14 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 ([676](#)), Directives ([676](#))

### **30.19.15 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 ([677](#))

### **30.19.16 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 ([677](#)) 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 ([677](#)), TSQLSCript.Script ([677](#)), TSQLSCript.Directives ([676](#))

### **30.19.17 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 ([676](#)) 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 ([676](#)), TSQLScript.Directives ([676](#))

### 30.19.18 TSQLScript.UseDefines

**Synopsis:** Automatically handle pre-processor defines

**Declaration:** Property UseDefines :

**Visibility:** published

**Access:**

**Description:** UseDefines will automatically register the following pre-processing directives:

```
#IFDEF
#IFNDEF
#ELSE
#ENDIF
#DEFINE
#UNDEF
#UNDEFINE
```

Additionally, these directives will be automatically handled by the TSQLScript component. This can be used to add conditional execution of the SQL script: they are treated as the conditional compilation statements found in the C macro preprocessor or the FPC conditional compilation features. The initial list of defined macros can be specified in the Defines (677) 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 (677), TSQLScript.Defines (677)

### 30.19.19 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 (675)

## 30.20 TSQLStatement

### 30.20.1 Description

TSQLStatement is a descendent of TCustomSQLStatement (648) which simply publishes the protected properties of that component.

See also: TCustomSQLStatement (648)

### 30.20.2 Property overview

Page	Property	Access	Description
680	Database		Database instance to execute statement on.
680	DataSource		Datasource to copy parameter values from
681	ParamCheck		Should SQL be checked for parameters
681	Params		List of parameters.
681	ParseSQL		Parse the SQL statement
681	SQL		The SQL statement to execute
682	Transaction		The transaction in which the SQL statement should be executed.

### 30.20.3 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 (651) descendent. It must be set, together with Transaction (630) in order to be able to call Prepare (649) or Execute (649).

See also: Transaction (630), Prepare (649), Execute (649)

### 30.20.4 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 (322) instance. When Execute (649) 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 (322), Execute (649), #fcl.db.TParam.Bound (403)

### 30.20.5 TSQLStatement.ParamCheck

**Synopsis:** Should SQL be checked for parameters

**Declaration:** Property ParamCheck :

Visibility: published

Access:

**Description:** ParamCheck must be set to False to disable the parameter check. The default value True indicates that the SQL statement should be checked for parameter names (in the form :ParamName), and corresponding TParam (395) instances should be added to the Params (630) 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 (395), Params (630), TSQLQuery.ParamCheck (671)

### 30.20.6 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 (630) statement (in the form :ParamName). The collection is filled automatically if the ParamCheck (630) property is True.

See also: SQL (630), ParamCheck (630), ParseSQL (630)

### 30.20.7 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 (630) 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 (630), ParamCheck (630)

### 30.20.8 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 (649) is called.

If ParamCheck (630) 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 (630) collection. If not, it will be emulated. The Params collection is automatically populated when the SQL statement is set.

Some databases support executing multiple SQL statements in 1 call. Therefor, no attempt is done to ensure that SQL contains a single SQL statement. However, error reporting and the RowsAffected (650) function may be wrong in such a case.

See also: ParseSQL (630), CheckParams (630), Params (630), Prepare (649), RowsAffected (650)

### 30.20.9 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 (630) property. This must be set before Prepare (649) is called.

See also: Database (630), Prepare (649), TSQLTransaction (682)

## 30.21 TSQLTransaction

### 30.21.1 Description

TSQLTransaction represents the transaction in which one or more TSQLQuery (662) 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 (651) used in an application.

See also: TSQLQuery (662), TSQLConnection (651)

### 30.21.2 Method overview

Page	Property	Description
683	Commit	Commit the transaction, end transaction context.
683	CommitRetaining	Commit the transaction, retain transaction context.
685	Create	Create a new transaction
685	Destroy	Destroy transaction component
685	EndTransaction	End the transaction
683	Rollback	Roll back all changes made in the current transaction.
684	RollbackRetaining	Roll back changes made in the transaction, keep transaction context.
684	StartTransaction	Start a new transaction

### 30.21.3 Property overview

Page	Property	Access	Description
<a href="#">685</a>	Action	rw	Currently unused in SQLDB
<a href="#">686</a>	Database		Database for which this component is handling connections
<a href="#">685</a>	Handle	r	Low-level transaction handle
<a href="#">686</a>	Params	rw	Transaction parameters

### 30.21.4 TSQLTransaction.Commit

**Synopsis:** Commit the transaction, end transaction context.

**Declaration:** procedure Commit; Virtual

**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 ([684](#)) method. To commit data while retaining an active transaction, execute CommitRetaining ([683](#)) instead.

**Errors:** Executing Commit when no transaction is active will result in an exception. A transaction must be started by calling StartTransaction ([684](#)). If the database backend reports an error, an exception is raised as well.

**See also:** StartTransaction ([684](#)), CommitRetaining ([683](#)), Rollback ([683](#)), RollbackRetaining ([684](#))

### 30.21.5 TSQLTransaction.CommitRetaining

**Synopsis:** Commit the transaction, retain transaction context.

**Declaration:** procedure CommitRetaining; Virtual

**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 ([683](#)) instead.

**Errors:** Executing CommitRetaining when no transaction is active will result in an exception. A transaction must be started by calling StartTransaction ([684](#)). If the database backend reports an error, an exception is raised as well.

**See also:** StartTransaction ([684](#)), Retaining ([683](#)), Rollback ([683](#)), RollbackRetaining ([684](#))

### 30.21.6 TSQLTransaction.Rollback

**Synopsis:** Roll back all changes made in the current transaction.

**Declaration:** procedure Rollback; Virtual

**Visibility:** public

**Description:** Rollback undoes all changes in the databack 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` (684) 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` (684). If the database backend reports an error, an exception is raised as well.

**See also:** `StartTransaction` (684), `CommitRetaining` (683), `Commit` (683), `RollbackRetaining` (684)

### 30.21.7 TSQLTransaction.RollbackRetaining

**Synopsis:** Roll back changes made in the transaction, keep transaction context.

**Declaration:** `procedure RollbackRetaining; Virtual`

**Visibility:** `public`

**Description:** `RollbackRetaining` undoes all changes in the databack 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` (683) instead.

**Remark:** Changes posted in datasets that are coupled to this transaction will not be undone in memory: these datasets must be reloaded from the database (using `Close` and `Open` to reload the data as it is in the database).

**Errors:** Executing `RollbackRetaining` when no transaction is active will result in an exception. A transaction must be started by calling `StartTransaction` (684). If the database backend reports an error, an exception is raised as well.

**See also:** `StartTransaction` (684), `Commit` (683), `Rollback` (683), `CommitRetaining` (683)

### 30.21.8 TSQLTransaction.StartTransaction

**Synopsis:** Start a new transaction

**Declaration:** `procedure StartTransaction; Override`

**Visibility:** `public`

**Description:** `StartTransaction` starts a new transaction context. All changes written to the database must be confirmed with a `Commit` (683) or can be undone with a `Rollback` (683) call.

Calling `StartTransaction` is equivalent to setting `Active` to `True`.

**Errors:** If `StartTransaction` is called while the transaction is still active, an exception will be raised.

**See also:** `StartTransaction` (684), `Commit` (683), `Rollback` (683), `CommitRetaining` (683), `EndTransaction` (685)

### 30.21.9 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.

### 30.21.10 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.

### 30.21.11 TSQLTransaction.EndTransaction

**Synopsis:** End the transaction

**Declaration:** procedure EndTransaction; Override

**Visibility:** public

**Description:** EndTransaction is equivalent to RollBack ([683](#)).

**See also:** RollBack ([683](#))

### 30.21.12 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 ([651](#)) descendent.

### 30.21.13 TSQLTransaction.Action

**Synopsis:** Currently unused in SQLDB

**Declaration:** Property Action : TCommitRollbackAction

**Visibility:** published

**Access:** Read,Write

**Description:** Action is currently unused in SQLDB.

### 30.21.14 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 (651) instance this transaction is handling transactions in. All datasets connected to this transaction component must have the same value for their Database (668) property.

**See also:** TSQLQuery.Database (668), TSQLConnection (651)

### 30.21.15 TSQLTransaction.Params

**Synopsis:** Transaction parameters

**Declaration:** Property Params : TStringList

**Visibility:** published

**Access:** Read,Write

**Description:** Params can be used to set connection-specific parameters in the form of Key=Value pairs. The contents of this property therefor depends on the type of connection.

**See also:** TSQLConnection (651)

# Chapter 31

## Reference for unit 'streamcoll'

### 31.1 Used units

Table 31.1: Used units by unit 'streamcoll'

Name	Page
Classes	??
System	??
sysutils	??

### 31.2 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` (690) 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.

### 31.3 Procedures and functions

#### 31.3.1 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` (689) and returns the read value. The value cannot be read and written across systems that have different endian values.

See also: [ColReadDateTime](#) (688), [ColWriteBoolean](#) (689), [ColReadString](#) (689), [ColReadInteger](#) (688), [ColReadFloat](#) (688), [ColReadCurrency](#) (688)

### **31.3.2 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` (689) and returns the read value. The value cannot be read and written across systems that have different endian values.

See also: [ColReadDateTime](#) (688), [ColReadBoolean](#) (687), [ColReadString](#) (689), [ColReadInteger](#) (688), [ColReadFloat](#) (688), [ColWriteCurrency](#) (689)

### **31.3.3 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` (689) and returns the read value. The value cannot be read and written across systems that have different endian values.

See also: [ColWriteDateTime](#) (689), [ColReadBoolean](#) (687), [ColReadString](#) (689), [ColReadInteger](#) (688), [ColReadFloat](#) (688), [ColReadCurrency](#) (688)

### **31.3.4 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` (690) and returns the read value. The value cannot be read and written across systems that have different endian values.

See also: [ColReadDateTime](#) (688), [ColReadBoolean](#) (687), [ColReadString](#) (689), [ColReadInteger](#) (688), [ColWriteFloat](#) (690), [ColReadCurrency](#) (688)

### **31.3.5 ColReadInteger**

Synopsis: Read a 32-bit integer from a stream.

Declaration: `function ColReadInteger(S: TStream) : Integer`

Visibility: default

**Description:** ColReadInteger reads a 32-bit integer from the stream S as it was written by ColWriteInteger (690) and returns the read value. The value cannot be read and written across systems that have different endian values.

**See also:** ColReadDateTime (688), ColReadBoolean (687), ColReadString (689), ColWriteInteger (690), ColReadFloat (688), ColReadCurrency (688)

### **31.3.6 ColReadString**

**Synopsis:** Read a string from a stream

**Declaration:** function ColReadString(S: TStream) : string

**Visibility:** default

**Description:** ColReadString reads a string value from the stream S as it was written by ColWriteString (690) and returns the read value. The value cannot be read and written across systems that have different endian values.

**See also:** ColReadDateTime (688), ColReadBoolean (687), ColWriteString (690), ColReadInteger (688), ColReadFloat (688), ColReadCurrency (688)

### **31.3.7 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 (687), ColWriteString (690), ColWriteInteger (690), ColWriteCurrency (689), ColWriteDateTime (689), ColWriteFloat (690)

### **31.3.8 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 (689), ColWriteString (690), ColWriteInteger (690), ColWriteDateTime (689), ColWriteFloat (690), ColReadCurrency (688)

### **31.3.9 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 (688), ColWriteBoolean (689), ColWriteString (690), ColWriteInteger (690), ColWriteFloat (690), ColWriteCurrency (689)

### **31.3.10 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 (689), ColWriteBoolean (689), ColWriteString (690), ColWriteInteger (690), ColReadFloat (688), ColWriteCurrency (689)

### **31.3.11 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 (689), ColWriteString (690), ColReadInteger (688), ColWriteCurrency (689), ColWriteDateTime (689)

### **31.3.12 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 (689), ColReadString (689), ColWriteInteger (690), ColWriteCurrency (689), ColWriteDateTime (689), ColWriteFloat (690)

## **31.4 EStreamColl**

### **31.4.1 Description**

Exception raised when an error occurs when streaming the collection.

## **31.5 TStreamCollection**

### **31.5.1 Description**

TStreamCollection is a TCollection (??) descendent which implements 2 calls LoadFromStream (691) and SaveToStream (691) which load and save the contents of the collection to a stream.

The collection items must be descendants of the TStreamCollectionItem (692) class for the streaming to work correctly.

Note that the stream must be used to load collections of the same type.

**See also:** TStreamCollectionItem (692)

### 31.5.2 Method overview

Page	Property	Description
<a href="#">691</a>	LoadFromStream	Load the collection from a stream
<a href="#">691</a>	SaveToStream	Load the collection from the stream.

### 31.5.3 Property overview

Page	Property	Access	Description
<a href="#">691</a>	Streaming	r	Indicates whether the collection is currently being written to stream

### 31.5.4 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 ([691](#)). 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 ([691](#))

### 31.5.5 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 ([691](#)). 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 ([691](#))

### 31.5.6 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 ([691](#)), TStreamCollection.SaveToStream ([691](#))

## **31.6 TStreamCollectionItem**

### **31.6.1 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 (690) to save or load the item from the stream.

See also: TStreamCollection (690)

# Chapter 32

## Reference for unit 'streamex'

### 32.1 Used units

Table 32.1: Used units by unit 'streamex'

Name	Page
Classes	??
System	??

### 32.2 Overview

streamex implements some extensions to be used together with streams from the classes unit.

### 32.3 TBidirBinaryObjectReader

#### 32.3.1 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 (694), TDelphiReader (694)

#### 32.3.2 Property overview

Page	Property	Access	Description
693	Position	rw	Position in the stream

#### 32.3.3 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 ([694](#)) class.

See also: TDelphiReader ([694](#))

## 32.4 TBidirBinaryObjectWriter

### 32.4.1 Description

TBidirBinaryObjectWriter is a class descendent from TBinaryObjectWriter (??), which implements the necessary support for BiDi data.

See also: TBidirBinaryObjectWriter ([694](#)), TDelphiWriter ([696](#))

### 32.4.2 Property overview

Page	Property	Access	Description
<a href="#">694</a>	Position	rw	Position in the stream

### 32.4.3 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 ([696](#)) class.

See also: TDelphiWriter ([696](#))

## 32.5 TDelphiReader

### 32.5.1 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 ([693](#)) driver class.

See also: TDelphiWriter ([696](#)), TBidirBinaryObjectReader ([693](#))

### 32.5.2 Method overview

Page	Property	Description
<a href="#">695</a>	GetDriver	Return the driver class as a TBidirBinaryObjectReader ( <a href="#">693</a> ) class
<a href="#">695</a>	Read	Read data from stream
<a href="#">695</a>	ReadStr	Overrides the standard ReadStr method

### 32.5.3 Property overview

Page	Property	Access	Description
<a href="#">695</a>	Position	rw	Position in the stream

### 32.5.4 TDelphiReader.GetDriver

Synopsis: Return the driver class as a TBidirBinaryObjectReader ([693](#)) class

Declaration: `function GetDriver : TBidirBinaryObjectReader`

Visibility: public

Description: `GetDriver` simply returns the used driver and typecasts it as `TBidirBinaryObjectReader` ([693](#)) class.

See also: `TBidirBinaryObjectReader` ([693](#))

### 32.5.5 TDelphiReader.ReadStr

Synopsis: Overrides the standard `ReadStr` method

Declaration: `function ReadStr : string`

Visibility: public

Description: `ReadStr` makes sure the `TBidirBinaryObjectReader` ([693](#)) methods are used, to store additional information about the stream position when reading the strings.

See also: `TBidirBinaryObjectReader` ([693](#))

### 32.5.6 TDelphiReader.Read

Synopsis: Read data from stream

Declaration: `procedure Read(var Buf; Count: LongInt); Override`

Visibility: public

Description: `Read` reads raw data from the stream. It reads `Count` bytes from the stream and places them in `Buf`. It forces the use of the `TBidirBinaryObjectReader` ([693](#)) class when reading.

See also: `TBidirBinaryObjectReader` ([693](#)), `TDelphiReader.Position` ([695](#))

### 32.5.7 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` ([695](#))

## 32.6 TDelphiWriter

### 32.6.1 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` (694) driver class.

See also: `TDelphiReader` (694), `TBidirBinaryObjectWriter` (694)

### 32.6.2 Method overview

Page	Property	Description
696	FlushBuffer	Flushes the stream buffer
696	GetDriver	Return the driver class as a <code>TBidirBinaryObjectWriter</code> (694) class
696	Write	Write raw data to the stream
697	WriteStr	Write a string to the stream
697	WriteValue	Write value type

### 32.6.3 Property overview

Page	Property	Access	Description
697	Position	rw	Position in the stream

### 32.6.4 TDelphiWriter.GetDriver

**Synopsis:** Return the driver class as a `TBidirBinaryObjectWriter` (694) class

**Declaration:** `function GetDriver : TBidirBinaryObjectWriter`

**Visibility:** public

**Description:** `GetDriver` simply returns the used driver and typecasts it as `TBidirBinaryObjectWriter` (694) class.

See also: `TBidirBinaryObjectWriter` (694)

### 32.6.5 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.

### 32.6.6 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.

### 32.6.7 **TDelphiWriter.WriteString**

Synopsis: Write a string to the stream

Declaration: procedure WriteString(const Value: string)

Visibility: public

Description: WriteString writes a string to the stream, forcing the use of the TBidirBinaryObjectWriter (694) class methods, which update the position of the stream.

See also: TBidirBinaryObjectWriter (694)

### 32.6.8 **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 (694) methods, which update the position of the stream.

See also: TBidirBinaryObjectWriter (694)

### 32.6.9 **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 (694) instance used when streaming.

See also: TBidirBinaryObjectWriter (694)

## 32.7 **TStreamHelper**

### 32.7.1 **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 (??)

### 32.7.2 Method overview

Page	Property	Description
700	ReadDWordBE	Read a DWord from the stream, big endian
698	ReadDWordLE	Read a DWord from the stream, little endian
700	ReadQWordBE	Read a QWord from the stream, big endian
698	ReadQWordLE	Read a QWord from the stream, little endian
699	ReadWordBE	Read a Word from the stream, big endian
698	ReadWordLE	Read a Word from the stream, little endian
700	WriteDWordBE	Write a DWord value, big endian
699	WriteDWordLE	Write a DWord value, little endian
701	WriteQWordBE	Write a QWord value, big endian
699	WriteQWordLE	Write a QWord value, little endian
700	WriteWordBE	Write a word value, big endian
699	WriteWordLE	Write a word value, little endian

### 32.7.3 TStreamHelper.ReadWordLE

Synopsis: Read a Word from the stream, little endian

Declaration: function ReadWordLE : Word

Visibility: default

Description: ReadWordLE reads a word from the stream, little-endian (LSB first).

Errors: If not enough data is available an EReadError exception is raised.

See also: ReadDWordLE (693), ReadQWordLE (693), WriteWordLE (693)

### 32.7.4 TStreamHelper.ReadDWordLE

Synopsis: Read a DWord from the stream, little endian

Declaration: function ReadDWordLE : dword

Visibility: default

Description: ReadWordLE reads a DWord from the stream, little-endian (LSB first).

Errors: If not enough data is available an EReadError exception is raised.

See also: ReadWordLE (693), ReadQWordLE (693), WriteDWordLE (693)

### 32.7.5 TStreamHelper.ReadQWordLE

Synopsis: Read a QWord from the stream, little endian

Declaration: function ReadQWordLE : QWord

Visibility: default

Description: ReadWordLE reads a QWord from the stream, little-endian (LSB first).

Errors: If not enough data is available an EReadError exception is raised.

See also: ReadWordLE (693), ReadDWordLE (693), WriteQWordLE (693)

### 32.7.6 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:** ReadWordLE ([693](#)), WriteDWordLE ([693](#)), WriteQWordLE ([693](#))

### 32.7.7 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:** ReadDWordLE ([693](#)), WriteWordLE ([693](#)), WriteQWordLE ([693](#))

### 32.7.8 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:** ReadQWordLE ([693](#)), WriteDWordLE ([693](#)), WriteWordLE ([693](#))

### 32.7.9 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:** ReadDWordBE ([693](#)), ReadQWordBE ([693](#)), WriteWordBE ([693](#))

### 32.7.10 **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:** ReadWordBE ([693](#)), ReadQWordBE ([693](#)), WriteDWordBE ([693](#))

### 32.7.11 **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:** ReadWordBE ([693](#)), ReadDWordBE ([693](#)), WriteQWordBE ([693](#))

### 32.7.12 **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:** ReadWordBE ([693](#)), WriteDWordBE ([693](#)), WriteQWordBE ([693](#))

### 32.7.13 **TStreamHelper.WriteDWordBE**

**Synopsis:** Write a DWord value, big endian

**Declaration:** procedure WriteDWordBE (dw: dword)

**Visibility:** default

**Description:** WriteDWordBE writes a DWord-sized value to the stream, big-endian (MSB first).

**Errors:** If not all data (4 bytes) can be written, an EWriteError exception is raised.

**See also:** ReadDWordBE ([693](#)), WriteWordBE ([693](#)), WriteQWordBE ([693](#))

### 32.7.14 TStreamHelper.WriteQWordBE

**Synopsis:** Write a QWord value, big endian

**Declaration:** procedure WriteQWordBE (dq: QWord)

**Visibility:** default

**Description:** WriteQWordBE writes a QWord-sized value to the stream, big-endian (MSB first).

**Errors:** If not all data (8 bytes) can be written, an EWriteError exception is raised.

**See also:** ReadQWordBE ([693](#)), WriteDWordBE ([693](#)), WriteWordBE ([693](#))

# Chapter 33

## Reference for unit 'StreamIO'

### 33.1 Used units

Table 33.1: Used units by unit 'StreamIO'

Name	Page
Classes	??
System	??
sysutils	??

### 33.2 Overview

The StreamIO unit implements a call to reroute the input or output of a text file to a descendants of TStream (??).

This allows to use the standard pascal Read (??) and Write (??) functions (with all their possibilities), on streams.

### 33.3 Procedures and functions

#### 33.3.1 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 ([703](#))

### **33.3.2 GetStream**

**Synopsis:** Return the stream, associated with a file.

**Declaration:** function GetStream(var F: TTTextRec) : TStream

**Visibility:** default

**Description:** GetStream returns the instance of the stream that was associated with the file F using AssignStream ([702](#)).

**Errors:** An invalid class reference will be returned if the file was not associated with a stream.

**See also:** AssignStream ([702](#)), TStream ([??](#))

# Chapter 34

## Reference for unit 'syncobjs'

### 34.1 Used units

Table 34.1: Used units by unit 'syncobjs'

Name	Page
System	??
sysutils	??

### 34.2 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` (705) class is a wrapper around low-level critical section routines (semaphores or mutexes). The `TEventObject` (707) class can be used to send messages between threads (also known as conditional variables in Posix threads).

### 34.3 Constants, types and variables

#### 34.3.1 Constants

`INFINITE = (-1)`

Constant denoting an infinite timeout.

#### 34.3.2 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` (707) class.

---

```
TEventHandle = Pointer
```

TEventHandle is an opaque type and should not be used in user code.

```
TWaitResult = (wrSignaled, wrTimeout, wrAbandoned, wrError)
```

Table 34.2: Enumeration values for type TWaitResult

Value	Explanation
wrAbandoned	Wait operation was abandoned.
wrError	An error occurred during the wait operation.
wrSignaled	Event was signaled (triggered)
wrTimeout	Time-out period expired

TWaitResult is used to report the result of a wait operation.

## 34.4 TCriticalSection

### 34.4.1 Description

TCriticalSection is a class wrapper around the low-level TRTLCriticalSection routines. It simply calls the RTL routines in the system unit for critical section support.

A critical section is a resource which can be owned by only 1 caller: it can be used to make sure that in a multithreaded application only 1 thread enters pieces of code protected by the critical section.

Typical usage is to protect a piece of code with the following code (MySection is a TCriticalSection instance):

```
// Previous code
MySection.Acquire;
Try
  // Protected code
Finally
  MySection.Release;
end;
// Other code.
```

The protected code can be executed by only 1 thread at a time. This is useful for instance for list operations in multithreaded environments.

See also: Acquire ([706](#)), Release ([706](#))

### 34.4.2 Method overview

Page	Property	Description
<a href="#">706</a>	Acquire	Enter the critical section
<a href="#">707</a>	Create	Create a new critical section.
<a href="#">707</a>	Destroy	Destroy the criticalsection instance
<a href="#">706</a>	Enter	Alias for Acquire
<a href="#">707</a>	Leave	Alias for Release
<a href="#">706</a>	Release	Leave the critical section
<a href="#">706</a>	TryEnter	Try and obtain the critical section

### 34.4.3 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 ([706](#))

### 34.4.4 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 ([706](#))

### 34.4.5 TCriticalSection.Enter

Synopsis: Alias for Acquire

Declaration: procedure Enter

Visibility: public

Description: Enter just calls Acquire ([706](#)).

See also: Leave ([707](#)), Acquire ([706](#))

### 34.4.6 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 ([706](#))

### 34.4.7 TCriticalSection.Leave

**Synopsis:** Alias for Release

**Declaration:** procedure Leave

**Visibility:** public

**Description:** Leave just calls Release ([706](#))

**See also:** Release ([706](#)), Enter ([706](#))

### 34.4.8 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 ([707](#))

### 34.4.9 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 ([707](#)), Acquire ([706](#))

## 34.5 TEventObject

### 34.5.1 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 ([705](#))

### 34.5.2 Method overview

Page	Property	Description
708	Create	Create a new event object
708	destroy	Clean up the event and release from memory
708	ResetEvent	Reset the event
709	SetEvent	Set the event
709	WaitFor	Wait for the event to be set.

### 34.5.3 Property overview

Page	Property	Access	Description
709	ManualReset	r	Should the event be reset manually

### 34.5.4 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 security attributes EventAttributes (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 (709), ResetEvent (708)

### 34.5.5 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 (708)

### 34.5.6 TEventObject.ResetEvent

Synopsis: Reset the event

Declaration: procedure ResetEvent

Visibility: public

Description: ResetEvent turns off the event. Any WaitFor (709) operation will suspend the calling thread.

See also: SetEvent (709), WaitFor (709)

### 34.5.7 TEventObject.SetEvent

**Synopsis:** Set the event

**Declaration:** procedure SetEvent

**Visibility:** public

**Description:** SetEvent sets the event. If the ManualReset ([709](#)) is True any thread that was waiting for the event to be set (using WaitFor ([709](#))) will resume its 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 ([709](#)), ManualReset ([709](#))

### 34.5.8 TEventObject.WaitFor

**Synopsis:** Wait for the event to be set.

**Declaration:** function WaitFor(Timeout: Cardinal) : TWaitResult

**Visibility:** public

**Description:** WaitFor should be used in threads that should be notified when the event is set. When WaitFor is called, and the event is not set, the thread will be suspended. As soon as the event is set by some other thread (using SetEvent ([709](#))) 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 ([709](#))

### 34.5.9 TEventObject.ManualReset

**Synopsis:** Should the event be reset manually

**Declaration:** Property ManualReset : Boolean

**Visibility:** public

**Access:** Read

**Description:** Should the event be reset manually

## 34.6 THandleObject

### 34.6.1 Description

THandleObject is a parent class for synchronization classes that need to store an operating system handle. It introduces a property Handle ([710](#)) which can be used to store the operating system handle. The handle is in no way manipulated by THandleObject, only storage is provided.

**See also:** Handle ([710](#))

### 34.6.2 Method overview

Page	Property	Description
<a href="#">710</a>	destroy	Free the instance

### 34.6.3 Property overview

Page	Property	Access	Description
<a href="#">710</a>	Handle	r	Handle for this object
<a href="#">710</a>	LastError	r	Last operating system error

### 34.6.4 THandleObject.destroy

**Synopsis:** Free the instance

**Declaration:** `destructor destroy; Override`

**Visibility:** public

**Description:** Destroy does nothing in the Free Pascal implementation of THandleObject.

### 34.6.5 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, descendant classes should set the handle by accessing it's protected field FHandle directly.

### 34.6.6 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 ([710](#)).

**See also:** Handle ([710](#))

## 34.7 TSimpleEvent

### 34.7.1 Description

TSimpleEvent is a simple descendent of the TEventObject ([707](#)) class. It creates an event with no name, which must be reset manually, and which is initially not set.

**See also:** TEventObject ([707](#)), TSsimpleEvent.Create ([711](#))

### 34.7.2 Method overview

Page	Property	Description
<a href="#">711</a>	Create	Creates a new TSimpleEvent instance

### 34.7.3 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 ([708](#)) with Nil for the security attributes, an empty name, AManualReset set to True, and InitialState to False.

See also: TEventObject.Create ([708](#))

## 34.8 TSynchroObject

### 34.8.1 Description

TSynchroObject is an abstract synchronization resource object. It implements 2 virtual methods Acquire ([711](#)) which can be used to acquire the resource, and Release ([711](#)) to release the resource.

See also: Acquire ([711](#)), Release ([711](#))

### 34.8.2 Method overview

Page	Property	Description
<a href="#">711</a>	Acquire	Acquire synchronization resource
<a href="#">711</a>	Release	Release previously acquired synchronization resource

### 34.8.3 TSynchroObject.Acquire

Synopsis: Acquire synchronization resource

Declaration: procedure Acquire; Virtual

Visibility: default

Description: Acquire does nothing in TSynchroObject. Descendent classes must override this method to acquire the resource they manage.

See also: Release ([711](#))

### 34.8.4 TSynchroObject.Release

Synopsis: Release previously acquired synchronization resource

Declaration: procedure Release; Virtual

Visibility: default

**Description:** Release does nothing in TSynchroObject. Descendent classes must override this method to release the resource they acquired through the Acquire ([711](#)) call.

**See also:** Acquire ([711](#))

# Chapter 35

## Reference for unit 'URIParser'

### 35.1 Overview

The URIParser unit contains a basic type (TURI (713)) and some routines for the parsing (ParseURI (714)) and construction (EncodeURI (713)) 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..

### 35.2 Constants, types and variables

#### 35.2.1 Types

```
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 (714) call. It contains the contents of a URI, parsed out in its various pieces.

### 35.3 Procedures and functions

#### 35.3.1 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 ([714](#))

### 35.3.2 FilenameToURI

Synopsis: Construct a URI from a filename

Declaration: function FilenameToURI(const Filename: string; Encode: Boolean) : string

Visibility: default

Description: FilenameToURI takes Filename and constructs a file: protocol URI from it.

Errors: None.

See also: URIToFilename ([715](#))

### 35.3.3 IsAbsoluteURI

Synopsis: Check whether a URI is absolute.

Declaration: function IsAbsoluteURI(const UriReference: string) : Boolean

Visibility: default

Description: IsAbsoluteURI returns True if the URI in UriReference is absolute, i.e. contains a protocol part.

Errors: None.

See also: FilenameToURI ([714](#)), URIToFileName ([715](#))

### 35.3.4 ParseURI

Synopsis: Parse a URI and split it into its constituent parts

Declaration: function ParseURI(const URI: string; Decode: Boolean) : TURI; Overload  
function ParseURI(const URI: string; const DefaultProtocol: string;  
DefaultPort: Word; Decode: Boolean) : TURI; Overload

Visibility: default

Description: ParseURI decodes URI and returns the various parts of the URI in the result record.

The function accepts the most general URI scheme:

```
proto://user:pwd@host:port/path/document?params#bookmark
```

Missing (optional) parts in the URI will be left blank in the result record. If a default protocol and port are specified, they will be used in the record if the corresponding part is not present in the URI.

See also: EncodeURI ([713](#))

### 35.3.5 ResolveRelativeURI

**Synopsis:** Return a relative link

**Declaration:** function ResolveRelativeURI(const BaseUri: WideString;  
                  const RelUri: WideString;  
                  out ResultUri: WideString) : Boolean  
                  ; Overload  
function ResolveRelativeURI(const BaseUri: UTF8String;  
                  const RelUri: UTF8String;  
                  out ResultUri: UTF8String) : 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

### 35.3.6 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 ([715](#)), FilenameToURI ([714](#))

# Chapter 36

## Reference for unit 'zipper'

### 36.1 Used units

Table 36.1: Used units by unit 'zipper'

Name	Page
BaseUnix	??
Classes	??
System	??
sysutils	??
zstream	738

### 36.2 Overview

zipper implements zip compression/decompression compatible with the popular .ZIP format. The zip file format is documented at <http://www.pkware.com/documents/casestudies/APPNOTE.TXT>. The Pascal conversion of the standard zlib library was implemented by Jacques Nomssi Nzali. It is used in the FCL to implement the (??) class.

### 36.3 Constants, types and variables

#### 36.3.1 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

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.

END\_OF\_CENTRAL\_DIR\_SIGNATURE = \$06054B50

Marker specifying end of directory within zip file

```
FIRSTENTRY = 257
```

```
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.

```
TABLESIZE = 8191
```

### 36.3.2 Types

```
BufPtr = PByte
```

```
Central_File_Header_Type = packed record
  Signature : LongInt;
  MadeBy_Version : Word;
  Extract_Version_Reqd : Word;
  Bit_Flag : Word;
  Compress_Method : Word;
  Last_Mod_Time : Word;
  Last_Mod_Date : Word;
  Crc32 : LongWord;
  Compressed_Size : LongWord;
  Uncompressed_Size : LongWord;
  Filename_Length : Word;
  Extra_Field_Length : Word;
  File_Comment_Length : Word;
  Starting_Disk_Num : Word;
  Internal_Attributes : Word;
  External_Attributes : LongWord;
  Local_Header_Offset : LongWord;
end
```

This record contains the structure for a file header within the central directory.

```
CodeArray = Array[0..TABLESIZE] of CodeRec
```

```
CodeRec = packed record
  Child : SmallInt;
  Sibling : SmallInt;
  Suffix : Byte;
end
```

```
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.

```
FreeListArray = Array[FIRSTENTRY..TABLESIZE] of Word
```

```
FreeListPtr = ^FreeListArray
```

```
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

```
TablePtr = ^CodeArray
```

```
TCustomInputStreamEvent = procedure(Sender: TObject;
  var AStream: TStream) of object
```

```
TOnCustomStreamEvent = procedure(Sender: TObject; var AStream: TStream;
  AItem: TFullZipFileEntry) of object
```

```
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

## 36.4 EZipError

### 36.4.1 Description

Exception specific to the zipper unit

## 36.5 TCompressor

### 36.5.1 Description

This object compresses a stream into a compressed zip stream.

### 36.5.2 Method overview

Page	Property	Description
720	Compress	Compresses input stream to output stream
719	Create	Creates a (??) object
720	ZipBitFlag	
720	ZipID	Identifier for type of compression
720	ZipVersionReqd	

### 36.5.3 Property overview

Page	Property	Access	Description
720	BufferSize	r	Size of buffer used for compression
721	Crc32Val	rw	Running CRC32 value
720	OnPercent	rw	Reference to OnPercent event handler
720	OnProgress	rw	Reference to OnProgress event handler

### 36.5.4 TCompressor.Create

Synopsis: Creates a (??) object

Declaration: constructor Create(AInFile: TStream; AOutFile: TStream;
ABufSize: LongWord); Virtual

Visibility: public

### **36.5.5 TCompressor.Compress**

**Synopsis:** Compresses input stream to output stream

**Declaration:** procedure Compress; Virtual; Abstract

**Visibility:** public

### **36.5.6 TCompressor.ZipID**

**Synopsis:** Identifier for type of compression

**Declaration:** class function ZipID; Virtual; Abstract

**Visibility:** public

### **36.5.7 TCompressor.ZipVersionReqd**

**Declaration:** class function ZipVersionReqd; Virtual; Abstract

**Visibility:** public

### **36.5.8 TCompressor.ZipBitFlag**

**Declaration:** function ZipBitFlag : Word; Virtual; Abstract

**Visibility:** public

### **36.5.9 TCompressor.BufferSize**

**Synopsis:** Size of buffer used for compression

**Declaration:** Property BufferSize : LongWord

**Visibility:** public

**Access:** Read

### **36.5.10 TCompressor.OnPercent**

**Synopsis:** Reference to OnPercent event handler

**Declaration:** Property OnPercent : Integer

**Visibility:** public

**Access:** Read,Write

### **36.5.11 TCompressor.OnProgress**

**Synopsis:** Reference to OnProgress event handler

**Declaration:** Property OnProgress : TProgressEvent

**Visibility:** public

**Access:** Read,Write

### 36.5.12 TCompressor.Crc32Val

**Synopsis:** Running CRC32 value

**Declaration:** Property Crc32Val : LongWord

**Visibility:** public

**Access:** Read,Write

**Description:** Running CRC32 value used when writing zip header

## 36.6 TDeCompressor

### 36.6.1 Description

This object decompresses a compressed zip stream.

### 36.6.2 Method overview

Page	Property	Description
<a href="#">721</a>	Create	Creates decompressor object
<a href="#">721</a>	DeCompress	Decompress zip stream
<a href="#">722</a>	ZipID	Identifier for type of compression

### 36.6.3 Property overview

Page	Property	Access	Description
<a href="#">722</a>	BufferSize	r	Size of buffer used in decompression
<a href="#">722</a>	Crc32Val	rw	Running CRC32 value used for verifying zip file integrity
<a href="#">722</a>	OnPercent	rw	Percentage of decompression completion
<a href="#">722</a>	OnProgress	rw	Event handler for OnProgress procedure

### 36.6.4 TDeCompressor.Create

**Synopsis:** Creates decompressor object

**Declaration:** constructor Create (AInFile: TStream; AOutFile: TStream;  
ABufSize: LongWord); Virtual

**Visibility:** public

### 36.6.5 TDeCompressor.DeCompress

**Synopsis:** Decompress zip stream

**Declaration:** procedure DeCompress; Virtual; Abstract

**Visibility:** public

### **36.6.6 TDeCompressor.ZipID**

Synopsis: Identifier for type of compression

Declaration: class function ZipID; Virtual; Abstract

Visibility: public

### **36.6.7 TDeCompressor.BufferSize**

Synopsis: Size of buffer used in decompression

Declaration: Property BufferSize : LongWord

Visibility: public

Access: Read

### **36.6.8 TDeCompressor.OnPercent**

Synopsis: Percentage of decompression completion

Declaration: Property OnPercent : Integer

Visibility: public

Access: Read,Write

### **36.6.9 TDeCompressor.OnProgress**

Synopsis: Event handler for OnProgress procedure

Declaration: Property OnProgress : TProgressEvent

Visibility: public

Access: Read,Write

### **36.6.10 TDeCompressor.Crc32Val**

Synopsis: Running CRC32 value used for verifying zip file integrity

Declaration: Property Crc32Val : LongWord

Visibility: public

Access: Read,Write

## **36.7 TDeflater**

### **36.7.1 Description**

Child of TCompressor ([719](#)) that implements the Deflate compression method

### 36.7.2 Method overview

Page	Property	Description
<a href="#">723</a>	Compress	
<a href="#">723</a>	Create	
<a href="#">723</a>	ZipBitFlag	
<a href="#">723</a>	ZipID	
<a href="#">723</a>	ZipVersionReqd	

### 36.7.3 Property overview

Page	Property	Access	Description
<a href="#">723</a>	CompressionLevel	rw	

### 36.7.4 TDeflater.Create

Declaration: constructor Create(AInFile: TStream; AOutFile: TStream;  
ABufSize: LongWord);   Override

Visibility: public

### 36.7.5 TDeflater.Compress

Declaration: procedure Compress;   Override

Visibility: public

### 36.7.6 TDeflater.ZipID

Declaration: class function ZipID;   Override

Visibility: public

### 36.7.7 TDeflater.ZipVersionReqd

Declaration: class function ZipVersionReqd;   Override

Visibility: public

### 36.7.8 TDeflater.ZipBitFlag

Declaration: function ZipBitFlag : Word;   Override

Visibility: public

### 36.7.9 TDeflater.CompressionLevel

Declaration: Property CompressionLevel : Tcompressionlevel

Visibility: public

Access: Read,Write

## 36.8 TFullZipFileEntries

### 36.8.1 Property overview

Page	Property	Access	Description
<a href="#">724</a>	FullEntries	rw	

### 36.8.2 TFullZipFileEntries.FullEntries

Declaration: Property FullEntries[AIndex: Integer]: TFullZipFileEntry; default

Visibility: public

Access: Read,Write

## 36.9 TFullZipFileEntry

### 36.9.1 Property overview

Page	Property	Access	Description
<a href="#">724</a>	CompressedSize	r	
<a href="#">724</a>	CompressMethod	r	
<a href="#">724</a>	CRC32	rw	

### 36.9.2 TFullZipFileEntry.CompressMethod

Declaration: Property CompressMethod : Word

Visibility: public

Access: Read

### 36.9.3 TFullZipFileEntry.CompressedSize

Declaration: Property CompressedSize : LongWord

Visibility: public

Access: Read

### 36.9.4 TFullZipFileEntry.CRC32

Declaration: Property CRC32 : LongWord

Visibility: public

Access: Read,Write

## 36.10 TInflater

### 36.10.1 Description

Child of TDeCompressor ([721](#)) that implements the Inflate decompression method

### 36.10.2 Method overview

Page	Property	Description
<a href="#">725</a>	Create	
<a href="#">725</a>	DeCompress	
<a href="#">725</a>	ZipID	

### 36.10.3 TInflater.Create

Declaration: constructor Create (AInFile: TStream; AOutFile: TStream;  
ABufSize: LongWord); Override

Visibility: public

### 36.10.4 TInflater.DeCompress

Declaration: procedure DeCompress; Override

Visibility: public

### 36.10.5 TInflater.ZipID

Declaration: class function ZipID; Override

Visibility: public

## 36.11 TShrinker

### 36.11.1 Description

Child of TCompressor ([719](#)) that implements the Shrink compression method

### 36.11.2 Method overview

Page	Property	Description
<a href="#">726</a>	Compress	
<a href="#">725</a>	Create	
<a href="#">726</a>	Destroy	
<a href="#">726</a>	ZipBitFlag	
<a href="#">726</a>	ZipID	
<a href="#">726</a>	ZipVersionReqd	

### 36.11.3 TShrinker.Create

Declaration: constructor Create (AInFile: TStream; AOutFile: TStream;  
ABufSize: LongWord); Override

Visibility: public

### 36.11.4 TShrinker.Destroy

Declaration: `destructor Destroy;    Override`

Visibility: `public`

### 36.11.5 TShrinker.Compress

Declaration: `procedure Compress;    Override`

Visibility: `public`

### 36.11.6 TShrinker.ZipID

Declaration: `class function ZipID;    Override`

Visibility: `public`

### 36.11.7 TShrinker.ZipVersionReqd

Declaration: `class function ZipVersionReqd;    Override`

Visibility: `public`

### 36.11.8 TShrinker.ZipBitFlag

Declaration: `function ZipBitFlag : Word;    Override`

Visibility: `public`

## 36.12 TUnZipper

### 36.12.1 Method overview

Page	Property	Description
<a href="#">728</a>	<code>Clear</code>	Removes all entries and files from object
<a href="#">727</a>	<code>Create</code>	
<a href="#">727</a>	<code>Destroy</code>	
<a href="#">728</a>	<code>Examine</code>	Opens zip file and reads the directory entries (list of zipped files)
<a href="#">727</a>	<code>UnZipAllFiles</code>	Unzips all files in a zip file, writing them to disk
<a href="#">728</a>	<code>UnZipFiles</code>	Unzips specified files

### 36.12.2 Property overview

Page	Property	Access	Description
728	BufferSize	rw	
730	Entries	r	
730	FileComment	r	
730	FileName	rw	Zip file to be unzipped/processed
730	Files	r	Files in zip file (deprecated)
728	OnCloseInputStream	rw	
729	OnCreateStream	rw	
729	OnDoneStream	rw	
729	OnEndFile	rw	Callback procedure that will be called after unzipping a file
728	OnOpenInputStream	rw	
729	OnPercent	rw	
729	OnProgress	rw	
729	OnStartFile	rw	Callback procedure that will be called before unzipping a file
730	OutputPath	rw	Path where archive files will be unzipped

### 36.12.3 TUnZipper.Create

Declaration: constructor Create

Visibility: public

### 36.12.4 TUnZipper.Destroy

Declaration: destructor Destroy; Override

Visibility: public

### 36.12.5 TUnZipper.UnZipAllFiles

Synopsis: Unzips all files in a zip file, writing them to disk

Declaration: procedure UnZipAllFiles; Virtual  
procedure UnZipAllFiles(AFileName: string)

Visibility: public

Description: This procedure unzips all files in a (??) 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.
```

### **36.12.6 TUnZipper.UnZipFiles**

**Synopsis:** Unzips specified files

**Declaration:** procedure UnZipFiles(AFileName: string; FileList: TStrings)  
procedure UnZipFiles(FileList: TStrings)

**Visibility:** public

### **36.12.7 TUnZipper.Clear**

**Synopsis:** Removes all entries and files from object

**Declaration:** procedure Clear

**Visibility:** public

### **36.12.8 TUnZipper.Examine**

**Synopsis:** Opens zip file and reads the directory entries (list of zipped files)

**Declaration:** procedure Examine

**Visibility:** public

### **36.12.9 TUnZipper.BufferSize**

**Declaration:** Property BufferSize : LongWord

**Visibility:** public

**Access:** Read,Write

### **36.12.10 TUnZipper.OnOpenInputStream**

**Declaration:** Property OnOpenInputStream : TCustomInputStreamEvent

**Visibility:** public

**Access:** Read,Write

### **36.12.11 TUnZipper.OnCloseInputStream**

**Declaration:** Property OnCloseInputStream : TCustomInputStreamEvent

**Visibility:** public

**Access:** Read,Write

### **36.12.12 TUnZipper.OnCreateStream**

**Declaration:** Property OnCreateStream : TOnCustomStreamEvent

**Visibility:** public

**Access:** Read,Write

### **36.12.13 TUnZipper.OnDoneStream**

**Declaration:** Property OnDoneStream : TOnCustomStreamEvent

**Visibility:** public

**Access:** Read,Write

### **36.12.14 TUnZipper.OnPercent**

**Declaration:** Property OnPercent : Integer

**Visibility:** public

**Access:** Read,Write

### **36.12.15 TUnZipper.OnProgress**

**Declaration:** Property OnProgress : TProgressEvent

**Visibility:** public

**Access:** Read,Write

### **36.12.16 TUnZipper.OnStartFile**

**Synopsis:** Callback procedure that will be called before unzipping a file

**Declaration:** Property OnStartFile : TOnStartFileEvent

**Visibility:** public

**Access:** Read,Write

### **36.12.17 TUnZipper.OnEndFile**

**Synopsis:** Callback procedure that will be called after unzipping a file

**Declaration:** Property OnEndFile : TOnEndOfFileEvent

**Visibility:** public

**Access:** Read,Write

### **36.12.18 TUnZipper.FileName**

Synopsis: Zip file to be unzipped/processed

Declaration: Property FileName : string

Visibility: public

Access: Read,Write

### **36.12.19 TUnZipper.OutputPath**

Synopsis: Path where archive files will be unzipped

Declaration: Property OutputPath : string

Visibility: public

Access: Read,Write

### **36.12.20 TUnZipper.FileComment**

Declaration: Property FileComment : string

Visibility: public

Access: Read

### **36.12.21 TUnZipper.Files**

Synopsis: Files in zip file (deprecated)

Declaration: Property Files : TStrings

Visibility: public

Access: Read

Description: List of files that should be compressed in the zip file. Note: deprecated. Use Entries.AddFileEntry(FileName) or Entries.AddFileEntries(List) instead.

### **36.12.22 TUnZipper.Entries**

Declaration: Property Entries : TFullZipFileEntries

Visibility: public

Access: Read

## **36.13 TZipFileEntries**

### **36.13.1 Description**

Files in the zip archive

### 36.13.2 Method overview

Page	Property	Description
<a href="#">731</a>	AddFileEntries	
<a href="#">731</a>	AddFileEntry	Adds file to zip directory

### 36.13.3 Property overview

Page	Property	Access	Description
<a href="#">731</a>	Entries	rw	Entries (files) in the zip archive

### 36.13.4 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: Adds a file to the list of files that will be written out in the zip file.

### 36.13.5 TZipFileEntries.AddFileEntries

Declaration: procedure AddFileEntries(const List: TStrings)

Visibility: public

### 36.13.6 TZipFileEntries.Entries

Synopsis: Entries (files) in the zip archive

Declaration: Property Entries[AIndex: Integer]: TZipFileEntry; default

Visibility: public

Access: Read,Write

## 36.14 TZipFileEntry

### 36.14.1 Method overview

Page	Property	Description
<a href="#">732</a>	Assign	
<a href="#">732</a>	Create	
<a href="#">732</a>	IsDirectory	
<a href="#">732</a>	IsLink	

### 36.14.2 Property overview

Page	Property	Access	Description
<a href="#">732</a>	ArchiveFileName	rw	
<a href="#">733</a>	Attributes	rw	
<a href="#">733</a>	CompressionLevel	rw	
<a href="#">733</a>	DateTime	rw	
<a href="#">733</a>	DiskFileName	rw	
<a href="#">733</a>	OS	rw	Indication of operating system/filesystem
<a href="#">733</a>	Size	rw	
<a href="#">732</a>	Stream	rw	

### 36.14.3 TZipFileEntry.Create

Declaration: constructor Create (ACollection: TCollection);   Override

Visibility: public

### 36.14.4 TZipFileEntry.IsDirectory

Declaration: function IsDirectory : Boolean

Visibility: public

### 36.14.5 TZipFileEntry.IsLink

Declaration: function IsLink : Boolean

Visibility: public

### 36.14.6 TZipFileEntry.Assign

Declaration: procedure Assign (Source: TPersistent);   Override

Visibility: public

### 36.14.7 TZipFileEntry.Stream

Declaration: Property Stream : TStream

Visibility: public

Access: Read,Write

### 36.14.8 TZipFileEntry.ArchiveFileName

Declaration: Property ArchiveFileName : string

Visibility: published

Access: Read,Write

### **36.14.9 TZipFileEntry.DiskFileName**

**Declaration:** Property DiskFileName : string

**Visibility:** published

**Access:** Read,Write

### **36.14.10 TZipFileEntry.Size**

**Declaration:** Property Size : Integer

**Visibility:** published

**Access:** Read,Write

### **36.14.11 TZipFileEntry.DateTime**

**Declaration:** Property DateTime : TDateTime

**Visibility:** published

**Access:** Read,Write

### **36.14.12 TZipFileEntry.OS**

**Synopsis:** Indication of operating system/filesystem

**Declaration:** Property OS : Byte

**Visibility:** published

**Access:** Read,Write

**Description:** Currently either OS\_UNIX (if UNIX is defined) or OS\_FAT.

### **36.14.13 TZipFileEntry.Attributes**

**Declaration:** Property Attributes : LongInt

**Visibility:** published

**Access:** Read,Write

### **36.14.14 TZipFileEntry.CompressionLevel**

**Declaration:** Property CompressionLevel : Tcompressionlevel

**Visibility:** published

**Access:** Read,Write

## 36.15 TZipper

### 36.15.1 Method overview

Page	Property	Description
735	Clear	
734	Create	
734	Destroy	
735	SaveToFile	
735	SaveToStream	
734	ZipAllFiles	Zips all files in object and writes zip to disk
735	ZipFiles	

### 36.15.2 Property overview

Page	Property	Access	Description
735	BufferSize	rw	
737	Entries	rw	
736	FileComment	rw	
736	FileName	rw	
736	Files	r	
736	InMemSize	rw	
736	OnEndFile	rw	
735	OnPercent	rw	
736	OnProgress	rw	
736	OnStartFile	rw	

### 36.15.3 TZipper.Create

Declaration: constructor Create

Visibility: public

### 36.15.4 TZipper.Destroy

Declaration: destructor Destroy; Override

Visibility: public

### 36.15.5 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 (734) object and writes the resulting zip file to disk.

An example of using this procedure:

```
uses
  Zipper;
var
```

```
Zipper: TZipper;
begin
  try
    Zipper := TZipper.Create;
    Zipper.FileName := ParamStr(1); //Use the first parameter on the command line as
    for I := 2 to ParamCount do //Use the other arguments on the command line as file
      Zipper.Entries.AddFileEntry(ParamStr(I), ParamStr(I));
    Zipper.ZipAllFiles;
  finally
    Zipper.Free;
  end;
end.
```

### 36.15.6 TZipper.SaveToFile

Declaration: procedure SaveToFile(AFileName: string)

Visibility: public

### 36.15.7 TZipper.SaveToStream

Declaration: procedure SaveToStream(AStream: TStream)

Visibility: public

### 36.15.8 TZipper.ZipFiles

Declaration: procedure ZipFiles(AFileName: string; FileList: TStrings)
 procedure ZipFiles(FileList: TStrings)
 procedure ZipFiles(AFileName: string; Entries: TZipFileEntries)
 procedure ZipFiles(Entries: TZipFileEntries)

Visibility: public

### 36.15.9 TZipper.Clear

Declaration: procedure Clear

Visibility: public

### 36.15.10 TZipper.BufferSize

Declaration: Property BufferSize : LongWord

Visibility: public

Access: Read,Write

### 36.15.11 TZipper.OnPercent

Declaration: Property OnPercent : Integer

Visibility: public

Access: Read,Write

### **36.15.12 TZipper.OnProgress**

**Declaration:** Property OnProgress : TProgressEvent

**Visibility:** public

**Access:** Read,Write

### **36.15.13 TZipper.OnStartFile**

**Declaration:** Property OnStartFile : TOnStartFileEvent

**Visibility:** public

**Access:** Read,Write

### **36.15.14 TZipper.OnEndFile**

**Declaration:** Property OnEndFile : TOnEndOfFileEvent

**Visibility:** public

**Access:** Read,Write

### **36.15.15 TZipper.FileName**

**Declaration:** Property FileName : string

**Visibility:** public

**Access:** Read,Write

### **36.15.16 TZipper.FileComment**

**Declaration:** Property FileComment : string

**Visibility:** public

**Access:** Read,Write

### **36.15.17 TZipper.Files**

**Declaration:** Property Files : TStrings; deprecated;

**Visibility:** public

**Access:** Read

### **36.15.18 TZipper.InMemSize**

**Declaration:** Property InMemSize : Integer

**Visibility:** public

**Access:** Read,Write

### **36.15.19 TZipper.Entries**

**Declaration:** Property Entries : TZipFileEntries

**Visibility:** public

**Access:** Read,Write

# Chapter 37

## Reference for unit 'zstream'

### 37.1 Used units

Table 37.1: Used units by unit 'zstream'

Name	Page
Classes	??
gzio	??
System	??
zbase	??

### 37.2 Overview

The ZStream unit implements a TStream (??) descendent (TCompressionStream (739)) 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 (742)) and decompresses it with the inflate algorithm.

### 37.3 Constants, types and variables

#### 37.3.1 Types

```
Tcompressionlevel = (clnone,clfastest,cldefault,clmax)
```

Table 37.2: Enumeration values for type Tcompressionlevel

Value	Explanation
cldefault	Use default compression
clfastest	Use fast (but less) compression.
clmax	Use maximum compression
clnone	Do not use compression, just copy data.

Compression level for the deflate algorithm

```
Tgzopenmode = (gzopenread, gzopenwrite)
```

Table 37.3: Enumeration values for type Tgzopenmode

Value	Explanation
gzopenread	Open file for reading
gzopenwrite	Open file for writing

Open mode for gzip file.

## 37.4 Ecompressionerror

### 37.4.1 Description

ECompressionError is the exception class used by the TCompressionStream ([739](#)) class.

## 37.5 Edecompressionerror

### 37.5.1 Description

EDecompressionError is the exception class used by the TDeCompressionStream ([742](#)) class.

## 37.6 Egzfileerror

### 37.6.1 Description

Egzfileerror is the exception class used to report errors by the Tgzfilestream ([745](#)) class.

See also: Tgzfilestream ([745](#))

## 37.7 Ezliberror

### 37.7.1 Description

Errors which occur in the zstream unit are signaled by raising an EZLibError exception descendant.

## 37.8 Tcompressionstream

### 37.8.1 Description

TCompressionStream

### 37.8.2 Method overview

Page	Property	Description
<a href="#">740</a>	create	Create a new instance of the compression stream.
<a href="#">740</a>	destroy	Flushes data to the output stream and destroys the compression stream.
<a href="#">741</a>	flush	Flush remaining data to the target stream
<a href="#">741</a>	get\_compressionrate	Get the current compression rate
<a href="#">740</a>	write	Write data to the stream

### 37.8.3 Property overview

Page	Property	Access	Description
<a href="#">741</a>	OnProgress		Progress handler

### 37.8.4 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 ([740](#))

### 37.8.5 Tcompressionstream.destroy

Synopsis: Flushe 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 ([740](#))

### 37.8.6 Tcompressionstream.write

Synopsis: Write data to the stream

Declaration: function write(const buffer; count: LongInt) : LongInt; Override

Visibility: public

**Description:** Write takes Count bytes from Buffer and compresses (deflates) them. The compressed result is written to the output stream.

**Errors:** If an error occurs, an ECompressionError ([739](#)) exception is raised.

**See also:** Read ([739](#)), Seek ([739](#))

### **37.8.7 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.

### **37.8.8 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.

### **37.8.9 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.

## **37.9 Tcustomzlibstream**

### **37.9.1 Description**

TCustomZlibStream serves as the ancestor class for the TCompressionStream ([739](#)) and TDecompressionStream ([742](#)) classes.

It introduces support for a progress handler, and stores the input or output stream.

### 37.9.2 Method overview

Page	Property	Description
<a href="#">742</a>	create	Create a new instance of TCustomZlibStream
<a href="#">742</a>	destroy	Clear up instance

### 37.9.3 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 descendants.

**See also:** [TCompressionStream \(739\)](#), [TDecompressionStream \(742\)](#)

### 37.9.4 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 \(742\)](#)

## 37.10 Tdecompressionstream

### 37.10.1 Description

TDecompressionStream performs the inverse operation of TCompressionStream ([739](#)). A read operation reads data from an input stream and decompresses (inflates) the data it as it goes along.

The decompression stream reads its compressed data from a stream with deflated data. This data can be created e.g. with a TCompressionStream ([739](#)) compression stream.

**See also:** [TCompressionStream \(739\)](#)

### 37.10.2 Method overview

Page	Property	Description
<a href="#">743</a>	create	Creates a new instance of the TDecompressionStream stream
<a href="#">743</a>	destroy	Destroys the TDecompressionStream instance
<a href="#">744</a>	get\_compressionrate	Get the current compression rate
<a href="#">743</a>	read	Read data from the compressed stream
<a href="#">744</a>	seek	Move stream position to a certain location in the stream.

### 37.10.3 Property overview

Page	Property	Access	Description
<a href="#">744</a>	OnProgress		Progress handler

### 37.10.4 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 ([743](#))

### 37.10.5 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 ([743](#))

### 37.10.6 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 ([739](#)) exception is raised.

**See also:** Write ([740](#))

### 37.10.7 Tdecompressionstream.seek

**Synopsis:** Move stream position to a certain location in the stream.

**Declaration:** function seek(offset: LongInt; origin: Word) : LongInt; Override

**Visibility:** public

**Description:** Seek overrides the standard Seek implementation. There are a few differences between the implementation of Seek in Free Pascal compared to Delphi:

- In Free Pascal, you can perform any seek. In case of a forward seek, the Free Pascal implementation will read some bytes until the desired position is reached, in case of a backward seek it will seek the source stream backwards to the position it had at the creation time of the TDecompressionStream and then again read some bytes until the desired position has been reached.
- In Free Pascal, a seek with soFromBeginning will reset the source stream to the position it had when the TDecompressionStream was created. In Delphi, the source stream is reset to position 0. This means that at creation time the source stream must always be at the start of the zstream, you cannot use TDecompressionStream.Seek to reset the source stream to the begin of the file.

**Errors:** An EDecompressionError ([739](#)) exception is raised if the stream does not allow the requested seek operation.

**See also:** Read ([743](#))

### 37.10.8 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.

### 37.10.9 Tdecompressionstream.OnProgress

**Synopsis:** Progress handler

**Declaration:** Property OnProgress :

**Visibility:** public

**Access:**

**Description:** OnProgress is called whenever input data is read from the source stream. It can be used to update a progress bar or so. The Sender argument to the progress handler is the decompression stream instance.

## 37.11 TGZFileStream

### 37.11.1 Description

TGZFileStream can be used to read data from a gzip file, or to write data to a gzip file.

See also: TCompressionStream (739), TDeCompressionStream (742)

### 37.11.2 Method overview

Page	Property	Description
745	create	Create a new instance of TGZFileStream
746	destroy	Removes TGZFileStream instance
745	read	Read data from the compressed file
746	seek	Set the position in the compressed stream.
746	write	Write data to be compressed

### 37.11.3 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 (745) 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 (746) method will be stored in the file in compressed (deflated) form.

Errors: If the file is not found, an EZlibError (739) exception is raised.

See also: Destroy (746), TGZOpenMode (739)

### 37.11.4 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 (745), Write (746), Seek (746)

### 37.11.5 TGZFileStream.write

**Synopsis:** Write data to be compressed

**Declaration:** function write(const buffer;count: LongInt) : LongInt; Override

**Visibility:** public

**Description:** Write writes Count bytes from Buffer to the compressed file. The data is compressed as it is written, so ideally, less than Count bytes end up in the compressed file. Note that it is not possible to write to the stream if it was opened in read mode.

The function returns the number of (uncompressed) bytes that were actually written.

**Errors:** In case of an error, an EZlibError (739) exception is raised.

**See also:** Create (745), Read (745), Seek (746)

### 37.11.6 TGZFileStream.seek

**Synopsis:** Set the position in the compressed stream.

**Declaration:** function seek(offset: LongInt;origin: Word) : LongInt; Override

**Visibility:** public

**Description:** Seek sets the position to Offset bytes, starting from Origin. Not all combinations are possible, see TDecompressionStream.Seek (744) for a list of possibilities.

**Errors:** In case an impossible combination is asked, an EZlibError (739) exception is raised.

**See also:** TDecompressionStream.Seek (744)

### 37.11.7 TGZFileStream.destroy

**Synopsis:** Removes TGZFileStream instance

**Declaration:** destructor destroy; Override

**Visibility:** public

**Description:** Destroy closes the file and releases the TGZFileStream instance from memory.

**See also:** Create (745)