

Free Component Library (FCL):
Reference guide.

Reference guide for FCL units.
Document version 2.6
October 2012

Michaël Van Canneyt

Contents

0.1	Overview	57
1	Reference for unit 'ascii85'	58
1.1	Used units	58
1.2	Overview	58
1.3	Constants, types and variables	58
1.3.1	Types	58
1.4	TASCII85DecoderStream	59
1.4.1	Description	59
1.4.2	Method overview	59
1.4.3	Property overview	59
1.4.4	TASCII85DecoderStream.Create	59
1.4.5	TASCII85DecoderStream.Decode	60
1.4.6	TASCII85DecoderStream.Close	60
1.4.7	TASCII85DecoderStream.ClosedP	60
1.4.8	TASCII85DecoderStream.Destroy	60
1.4.9	TASCII85DecoderStream.Read	61
1.4.10	TASCII85DecoderStream.Seek	61
1.4.11	TASCII85DecoderStream.BExpectBoundary	61
1.5	TASCII85EncoderStream	61
1.5.1	Description	61
1.5.2	Method overview	62
1.5.3	Property overview	62
1.5.4	TASCII85EncoderStream.Create	62
1.5.5	TASCII85EncoderStream.Destroy	62
1.5.6	TASCII85EncoderStream.Write	62
1.5.7	TASCII85EncoderStream.Width	63
1.5.8	TASCII85EncoderStream.Boundary	63
1.6	TASCII85RingBuffer	63
1.6.1	Description	63
1.6.2	Method overview	63

1.6.3	Property overview	63
1.6.4	TASCII85RingBuffer.Write	64
1.6.5	TASCII85RingBuffer.Read	64
1.6.6	TASCII85RingBuffer.FillCount	64
1.6.7	TASCII85RingBuffer.Size	64
2	Reference for unit 'AVL_Tree'	65
2.1	Used units	65
2.2	Overview	65
2.3	TAVLTree	65
2.3.1	Description	65
2.3.2	Method overview	66
2.3.3	Property overview	66
2.3.4	TAVLTree.Find	66
2.3.5	TAVLTree.FindKey	67
2.3.6	TAVLTree.FindSuccessor	67
2.3.7	TAVLTree.FindPrecessor	67
2.3.8	TAVLTree.FindLowest	67
2.3.9	TAVLTree.FindHighest	68
2.3.10	TAVLTree.FindNearest	68
2.3.11	TAVLTree.FindPointer	68
2.3.12	TAVLTree.FindLeftMost	68
2.3.13	TAVLTree.FindRightMost	69
2.3.14	TAVLTree.FindLeftMostKey	69
2.3.15	TAVLTree.FindRightMostKey	69
2.3.16	TAVLTree.FindLeftMostSameKey	69
2.3.17	TAVLTree.FindRightMostSameKey	70
2.3.18	TAVLTree.Add	70
2.3.19	TAVLTree.Delete	70
2.3.20	TAVLTree.Remove	70
2.3.21	TAVLTree.RemovePointer	71
2.3.22	TAVLTree.MoveDataLeftMost	71
2.3.23	TAVLTree.MoveDataRightMost	71
2.3.24	TAVLTree.Clear	71
2.3.25	TAVLTree.FreeAndClear	72
2.3.26	TAVLTree.FreeAndDelete	72
2.3.27	TAVLTree.ConsistencyCheck	72
2.3.28	TAVLTree.WriteReportToStream	72
2.3.29	TAVLTree.ReportAsString	73
2.3.30	TAVLTree.SetNodeManager	73

2.3.31	TAVLTree.Create	73
2.3.32	TAVLTree.Destroy	73
2.3.33	TAVLTree.GetEnumerator	74
2.3.34	TAVLTree.OnCompare	74
2.3.35	TAVLTree.Count	74
2.4	TAVLTreeNode	74
2.4.1	Description	74
2.4.2	Method overview	74
2.4.3	TAVLTreeNode.Clear	75
2.4.4	TAVLTreeNode.TreeDepth	75
2.5	TAVLTreeNodeEnumerator	75
2.5.1	Description	75
2.5.2	Method overview	75
2.5.3	Property overview	75
2.5.4	TAVLTreeNodeEnumerator.Create	75
2.5.5	TAVLTreeNodeEnumerator.MoveNext	76
2.5.6	TAVLTreeNodeEnumerator.Current	76
2.6	TAVLTreeNodeMemManager	76
2.6.1	Description	76
2.6.2	Method overview	76
2.6.3	Property overview	76
2.6.4	TAVLTreeNodeMemManager.DisposeNode	77
2.6.5	TAVLTreeNodeMemManager.NewNode	77
2.6.6	TAVLTreeNodeMemManager.Clear	77
2.6.7	TAVLTreeNodeMemManager.Create	77
2.6.8	TAVLTreeNodeMemManager.Destroy	77
2.6.9	TAVLTreeNodeMemManager.MinimumFreeNode	78
2.6.10	TAVLTreeNodeMemManager.MaximumFreeNodeRatio	78
2.6.11	TAVLTreeNodeMemManager.Count	78
2.7	TBaseAVLTreeNodeManager	78
2.7.1	Description	78
2.7.2	Method overview	79
2.7.3	TBaseAVLTreeNodeManager.DisposeNode	79
2.7.4	TBaseAVLTreeNodeManager.NewNode	79
3	Reference for unit 'base64'	80
3.1	Used units	80
3.2	Overview	80
3.3	Constants, types and variables	80
3.3.1	Types	80

3.4	Procedures and functions	81
3.4.1	DecodeStringBase64	81
3.4.2	EncodeStringBase64	81
3.5	EBase64DecodingException	81
3.5.1	Description	81
3.6	TBase64DecodingStream	81
3.6.1	Description	81
3.6.2	Method overview	82
3.6.3	Property overview	82
3.6.4	TBase64DecodingStream.Create	82
3.6.5	TBase64DecodingStream.Reset	82
3.6.6	TBase64DecodingStream.Read	82
3.6.7	TBase64DecodingStream.Seek	83
3.6.8	TBase64DecodingStream.EOF	83
3.6.9	TBase64DecodingStream.Mode	83
3.7	TBase64EncodingStream	84
3.7.1	Description	84
3.7.2	Method overview	84
3.7.3	TBase64EncodingStream.Destroy	84
3.7.4	TBase64EncodingStream.Flush	84
3.7.5	TBase64EncodingStream.Write	85
3.7.6	TBase64EncodingStream.Seek	85
4	Reference for unit 'BlowFish'	86
4.1	Used units	86
4.2	Overview	86
4.3	Constants, types and variables	86
4.3.1	Constants	86
4.3.2	Types	86
4.4	EBlowFishError	87
4.4.1	Description	87
4.5	TBlowFish	87
4.5.1	Description	87
4.5.2	Method overview	87
4.5.3	TBlowFish.Create	87
4.5.4	TBlowFish.Encrypt	88
4.5.5	TBlowFish.Decrypt	88
4.6	TBlowFishDeCryptStream	88
4.6.1	Description	88
4.6.2	Method overview	88

4.6.3	TBlowFishDeCryptStream.Read	88
4.6.4	TBlowFishDeCryptStream.Seek	89
4.7	TBlowFishEncryptStream	89
4.7.1	Description	89
4.7.2	Method overview	89
4.7.3	TBlowFishEncryptStream.Destroy	89
4.7.4	TBlowFishEncryptStream.Write	90
4.7.5	TBlowFishEncryptStream.Seek	90
4.7.6	TBlowFishEncryptStream.Flush	90
4.8	TBlowFishStream	91
4.8.1	Description	91
4.8.2	Method overview	91
4.8.3	Property overview	91
4.8.4	TBlowFishStream.Create	91
4.8.5	TBlowFishStream.Destroy	91
4.8.6	TBlowFishStream.BlowFish	92
5	Reference for unit 'bufstream'	93
5.1	Used units	93
5.2	Overview	93
5.3	Constants, types and variables	93
5.3.1	Constants	93
5.4	TBufStream	93
5.4.1	Description	93
5.4.2	Method overview	94
5.4.3	Property overview	94
5.4.4	TBufStream.Create	94
5.4.5	TBufStream.Destroy	94
5.4.6	TBufStream.Buffer	95
5.4.7	TBufStream.Capacity	95
5.4.8	TBufStream.BufferPos	95
5.4.9	TBufStream.BufferSize	95
5.5	TReadBufStream	96
5.5.1	Description	96
5.5.2	Method overview	96
5.5.3	TReadBufStream.Seek	96
5.5.4	TReadBufStream.Read	96
5.6	TWriteBufStream	97
5.6.1	Description	97
5.6.2	Method overview	97

5.6.3	TWriteBufStream.Destroy	97
5.6.4	TWriteBufStream.Seek	97
5.6.5	TWriteBufStream.Write	97
6	Reference for unit 'CacheCls'	99
6.1	Used units	99
6.2	Overview	99
6.3	Constants, types and variables	99
6.3.1	Resource strings	99
6.3.2	Types	99
6.4	ECacheError	100
6.4.1	Description	100
6.5	TCache	100
6.5.1	Description	100
6.5.2	Method overview	101
6.5.3	Property overview	101
6.5.4	TCache.Create	101
6.5.5	TCache.Destroy	101
6.5.6	TCache.Add	101
6.5.7	TCache.AddNew	102
6.5.8	TCache.FindSlot	102
6.5.9	TCache.IndexOf	102
6.5.10	TCache.Remove	103
6.5.11	TCache.Data	103
6.5.12	TCache.MRUSlot	103
6.5.13	TCache.LRUSlot	104
6.5.14	TCache.SlotCount	104
6.5.15	TCache.Slots	104
6.5.16	TCache.OnIsDataEqual	104
6.5.17	TCache.OnFreeSlot	105
7	Reference for unit 'contnrs'	106
7.1	Used units	106
7.2	Overview	106
7.3	Constants, types and variables	106
7.3.1	Constants	106
7.3.2	Types	107
7.4	Procedures and functions	110
7.4.1	RSHash	110
7.5	EDuplicate	110
7.5.1	Description	110

7.6	EKeyNotFound	110
7.6.1	Description	110
7.7	TBucketList	110
7.7.1	Description	110
7.7.2	Method overview	111
7.7.3	TBucketList.Create	111
7.8	TClassList	111
7.8.1	Description	111
7.8.2	Method overview	111
7.8.3	Property overview	111
7.8.4	TClassList.Add	112
7.8.5	TClassList.Extract	112
7.8.6	TClassList.Remove	112
7.8.7	TClassList.IndexOf	112
7.8.8	TClassList.First	113
7.8.9	TClassList.Last	113
7.8.10	TClassList.Insert	113
7.8.11	TClassList.Items	113
7.9	TComponentList	114
7.9.1	Description	114
7.9.2	Method overview	114
7.9.3	Property overview	114
7.9.4	TComponentList.Destroy	114
7.9.5	TComponentList.Add	114
7.9.6	TComponentList.Extract	115
7.9.7	TComponentList.Remove	115
7.9.8	TComponentList.IndexOf	115
7.9.9	TComponentList.First	116
7.9.10	TComponentList.Last	116
7.9.11	TComponentList.Insert	116
7.9.12	TComponentList.Items	116
7.10	TCustomBucketList	117
7.10.1	Description	117
7.10.2	Method overview	117
7.10.3	Property overview	117
7.10.4	TCustomBucketList.Destroy	117
7.10.5	TCustomBucketList.Clear	117
7.10.6	TCustomBucketList.Add	118
7.10.7	TCustomBucketList.Assign	118
7.10.8	TCustomBucketList.Exists	118

7.10.9	TCustomBucketList.Find	118
7.10.10	TCustomBucketList.ForEach	119
7.10.11	TCustomBucketList.Remove	119
7.10.12	TCustomBucketList.Data	119
7.11	TFPCustomHashTable	119
7.11.1	Description	119
7.11.2	Method overview	120
7.11.3	Property overview	120
7.11.4	TFPCustomHashTable.Create	120
7.11.5	TFPCustomHashTable.CreateWith	121
7.11.6	TFPCustomHashTable.Destroy	121
7.11.7	TFPCustomHashTable.ChangeTableSize	121
7.11.8	TFPCustomHashTable.Clear	121
7.11.9	TFPCustomHashTable.Delete	122
7.11.10	TFPCustomHashTable.Find	122
7.11.11	TFPCustomHashTable.IsEmpty	122
7.11.12	TFPCustomHashTable.HashFunction	122
7.11.13	TFPCustomHashTable.Count	123
7.11.14	TFPCustomHashTable.HashTableSize	123
7.11.15	TFPCustomHashTable.HashTable	123
7.11.16	TFPCustomHashTable.VoidSlots	123
7.11.17	TFPCustomHashTable.LoadFactor	124
7.11.18	TFPCustomHashTable.AVGChainLen	124
7.11.19	TFPCustomHashTable.MaxChainLength	124
7.11.20	TFPCustomHashTable.NumberOfCollisions	124
7.11.21	TFPCustomHashTable.Density	125
7.12	TFPDataHashTable	125
7.12.1	Description	125
7.12.2	Method overview	125
7.12.3	Property overview	125
7.12.4	TFPDataHashTable.Add	125
7.12.5	TFPDataHashTable.Items	126
7.13	TFPHashList	126
7.13.1	Description	126
7.13.2	Method overview	127
7.13.3	Property overview	127
7.13.4	TFPHashList.Create	127
7.13.5	TFPHashList.Destroy	127
7.13.6	TFPHashList.Add	128
7.13.7	TFPHashList.Clear	128

7.13.8	TFPHashList.NameOfIndex	128
7.13.9	TFPHashList.HashOfIndex	128
7.13.10	TFPHashList.GetNextCollision	129
7.13.11	TFPHashList.Delete	129
7.13.12	TFPHashList.Error	129
7.13.13	TFPHashList.Expand	129
7.13.14	TFPHashList.Extract	130
7.13.15	TFPHashList.IndexOf	130
7.13.16	TFPHashList.Find	130
7.13.17	TFPHashList.FindIndexOf	130
7.13.18	TFPHashList.FindWithHash	131
7.13.19	TFPHashList.Rename	131
7.13.20	TFPHashList.Remove	131
7.13.21	TFPHashList.Pack	131
7.13.22	TFPHashList.ShowStatistics	132
7.13.23	TFPHashList.ForEachCall	132
7.13.24	TFPHashList.Capacity	132
7.13.25	TFPHashList.Count	132
7.13.26	TFPHashList.Items	133
7.13.27	TFPHashList.List	133
7.13.28	TFPHashList.Strs	133
7.14	TFPHashObject	133
7.14.1	Description	133
7.14.2	Method overview	134
7.14.3	Property overview	134
7.14.4	TFPHashObject.CreateNotOwned	134
7.14.5	TFPHashObject.Create	134
7.14.6	TFPHashObject.ChangeOwner	134
7.14.7	TFPHashObject.ChangeOwnerAndName	135
7.14.8	TFPHashObject.Rename	135
7.14.9	TFPHashObject.Name	135
7.14.10	TFPHashObject.Hash	135
7.15	TFPHashObjectList	136
7.15.1	Method overview	136
7.15.2	Property overview	136
7.15.3	TFPHashObjectList.Create	136
7.15.4	TFPHashObjectList.Destroy	136
7.15.5	TFPHashObjectList.Clear	137
7.15.6	TFPHashObjectList.Add	137
7.15.7	TFPHashObjectList.NameOfIndex	137

7.15.8	TFPHashObjectList.HashOfIndex	138
7.15.9	TFPHashObjectList.GetNextCollision	138
7.15.10	TFPHashObjectList.Delete	138
7.15.11	TFPHashObjectList.Expand	138
7.15.12	TFPHashObjectList.Extract	139
7.15.13	TFPHashObjectList.Remove	139
7.15.14	TFPHashObjectList.IndexOf	139
7.15.15	TFPHashObjectList.Find	139
7.15.16	TFPHashObjectList.FindIndexOf	140
7.15.17	TFPHashObjectList.FindWithHash	140
7.15.18	TFPHashObjectList.Rename	140
7.15.19	TFPHashObjectList.FindInstanceOf	140
7.15.20	TFPHashObjectList.Pack	141
7.15.21	TFPHashObjectList.ShowStatistics	141
7.15.22	TFPHashObjectList.ForEachCall	141
7.15.23	TFPHashObjectList.Capacity	141
7.15.24	TFPHashObjectList.Count	142
7.15.25	TFPHashObjectList.OwnsObjects	142
7.15.26	TFPHashObjectList.Items	142
7.15.27	TFPHashObjectList.List	142
7.16	TFPObjectHashTable	143
7.16.1	Description	143
7.16.2	Method overview	143
7.16.3	Property overview	143
7.16.4	TFPObjectHashTable.Create	143
7.16.5	TFPObjectHashTable.CreateWith	143
7.16.6	TFPObjectHashTable.Add	144
7.16.7	TFPObjectHashTable.Items	144
7.16.8	TFPObjectHashTable.OwnsObjects	144
7.17	TFPObjectList	145
7.17.1	Description	145
7.17.2	Method overview	145
7.17.3	Property overview	145
7.17.4	TFPObjectList.Create	145
7.17.5	TFPObjectList.Destroy	146
7.17.6	TFPObjectList.Clear	146
7.17.7	TFPObjectList.Add	146
7.17.8	TFPObjectList.Delete	147
7.17.9	TFPObjectList.Exchange	147
7.17.10	TFPObjectList.Expand	147

7.17.11 TFPObjectList.Extract	147
7.17.12 TFPObjectList.Remove	148
7.17.13 TFPObjectList.IndexOf	148
7.17.14 TFPObjectList.FindInstanceOf	148
7.17.15 TFPObjectList.Insert	149
7.17.16 TFPObjectList.First	149
7.17.17 TFPObjectList.Last	149
7.17.18 TFPObjectList.Move	149
7.17.19 TFPObjectList.Assign	150
7.17.20 TFPObjectList.Pack	150
7.17.21 TFPObjectList.Sort	150
7.17.22 TFPObjectList.ForEachCall	151
7.17.23 TFPObjectList.Capacity	151
7.17.24 TFPObjectList.Count	151
7.17.25 TFPObjectList.OwnsObjects	151
7.17.26 TFPObjectList.Items	152
7.17.27 TFPObjectList.List	152
7.18 TFPStringHashTable	152
7.18.1 Description	152
7.18.2 Method overview	152
7.18.3 Property overview	153
7.18.4 TFPStringHashTable.Add	153
7.18.5 TFPStringHashTable.Items	153
7.19 THTCustomNode	153
7.19.1 Description	153
7.19.2 Method overview	153
7.19.3 Property overview	153
7.19.4 THTCustomNode.CreateWith	154
7.19.5 THTCustomNode.HasKey	154
7.19.6 THTCustomNode.Key	154
7.20 THTDataNode	154
7.20.1 Description	154
7.20.2 Property overview	155
7.20.3 THTDataNode.Data	155
7.21 THTObjectNode	155
7.21.1 Description	155
7.21.2 Property overview	155
7.21.3 THTObjectNode.Data	155
7.22 THTOwnedObjectNode	155
7.22.1 Description	155

7.22.2	Method overview	156
7.22.3	THTOwnedObjectNode.Destroy	156
7.23	THTStringNode	156
7.23.1	Description	156
7.23.2	Property overview	156
7.23.3	THTStringNode.Data	156
7.24	TObjectBucketList	156
7.24.1	Description	156
7.24.2	Method overview	157
7.24.3	Property overview	157
7.24.4	TObjectBucketList.Add	157
7.24.5	TObjectBucketList.Remove	157
7.24.6	TObjectBucketList.Data	157
7.25	TObjectList	158
7.25.1	Description	158
7.25.2	Method overview	158
7.25.3	Property overview	158
7.25.4	TObjectList.create	158
7.25.5	TObjectList.Add	158
7.25.6	TObjectList.Extract	159
7.25.7	TObjectList.Remove	159
7.25.8	TObjectList.IndexOf	159
7.25.9	TObjectList.FindInstanceOf	160
7.25.10	TObjectList.Insert	160
7.25.11	TObjectList.First	160
7.25.12	TObjectList.Last	160
7.25.13	TObjectList.OwnsObjects	161
7.25.14	TObjectList.Items	161
7.26	TObjectQueue	161
7.26.1	Method overview	161
7.26.2	TObjectQueue.Push	161
7.26.3	TObjectQueue.Pop	162
7.26.4	TObjectQueue.Peek	162
7.27	TObjectStack	162
7.27.1	Description	162
7.27.2	Method overview	162
7.27.3	TObjectStack.Push	162
7.27.4	TObjectStack.Pop	163
7.27.5	TObjectStack.Peek	163
7.28	TOrderedList	163

7.28.1	Description	163
7.28.2	Method overview	164
7.28.3	TOrderedList.Create	164
7.28.4	TOrderedList.Destroy	164
7.28.5	TOrderedList.Count	164
7.28.6	TOrderedList.AtLeast	165
7.28.7	TOrderedList.Push	165
7.28.8	TOrderedList.Pop	165
7.28.9	TOrderedList.Peek	165
7.29	TQueue	166
7.29.1	Description	166
7.30	TStack	166
7.30.1	Description	166
8	Reference for unit 'CustApp'	167
8.1	Used units	167
8.2	Overview	167
8.3	Constants, types and variables	167
8.3.1	Types	167
8.3.2	Variables	168
8.4	TCustomApplication	168
8.4.1	Description	168
8.4.2	Method overview	168
8.4.3	Property overview	169
8.4.4	TCustomApplication.Create	169
8.4.5	TCustomApplication.Destroy	169
8.4.6	TCustomApplication.HandleException	169
8.4.7	TCustomApplication.Initialize	170
8.4.8	TCustomApplication.Run	170
8.4.9	TCustomApplication.ShowException	170
8.4.10	TCustomApplication.Terminate	171
8.4.11	TCustomApplication.FindOptionIndex	171
8.4.12	TCustomApplication.GetOptionValue	171
8.4.13	TCustomApplication.HasOption	172
8.4.14	TCustomApplication.CheckOptions	172
8.4.15	TCustomApplication.GetEnvironmentList	173
8.4.16	TCustomApplication.Log	173
8.4.17	TCustomApplication.ExeName	173
8.4.18	TCustomApplication.HelpFile	174
8.4.19	TCustomApplication.Terminated	174

8.4.20	TCustomApplication.Title	174
8.4.21	TCustomApplication.OnException	175
8.4.22	TCustomApplication.ConsoleApplication	175
8.4.23	TCustomApplication.Location	175
8.4.24	TCustomApplication.Params	175
8.4.25	TCustomApplication.ParamCount	176
8.4.26	TCustomApplication.EnvironmentVariable	176
8.4.27	TCustomApplication.OptionChar	176
8.4.28	TCustomApplication.CaseSensitiveOptions	177
8.4.29	TCustomApplication.StopOnException	177
8.4.30	TCustomApplication.EventLogFilter	177
9	Reference for unit 'daemonapp'	178
9.1	Used units	178
9.2	Overview	178
9.3	Daemon application architecture	179
9.4	Constants, types and variables	179
9.4.1	Resource strings	179
9.4.2	Types	180
9.4.3	Variables	183
9.5	Procedures and functions	183
9.5.1	Application	183
9.5.2	DaemonError	184
9.5.3	RegisterDaemonApplicationClass	184
9.5.4	RegisterDaemonClass	184
9.5.5	RegisterDaemonMapper	184
9.6	EDaemon	185
9.6.1	Description	185
9.7	TCustomDaemon	185
9.7.1	Description	185
9.7.2	Method overview	185
9.7.3	Property overview	185
9.7.4	TCustomDaemon.LogMessage	185
9.7.5	TCustomDaemon.ReportStatus	186
9.7.6	TCustomDaemon.Definition	186
9.7.7	TCustomDaemon.DaemonThread	186
9.7.8	TCustomDaemon.Controller	187
9.7.9	TCustomDaemon.Status	187
9.7.10	TCustomDaemon.Logger	187
9.8	TCustomDaemonApplication	187

9.8.1	Description	187
9.8.2	Method overview	188
9.8.3	Property overview	188
9.8.4	TCustomDaemonApplication.Destroy	188
9.8.5	TCustomDaemonApplication.ShowException	188
9.8.6	TCustomDaemonApplication.CreateDaemon	188
9.8.7	TCustomDaemonApplication.StopDaemons	189
9.8.8	TCustomDaemonApplication.InstallDaemons	189
9.8.9	TCustomDaemonApplication.RunDaemons	189
9.8.10	TCustomDaemonApplication.UnInstallDaemons	189
9.8.11	TCustomDaemonApplication.ShowHelp	190
9.8.12	TCustomDaemonApplication.CreateForm	190
9.8.13	TCustomDaemonApplication.OnRun	190
9.8.14	TCustomDaemonApplication.EventLog	191
9.8.15	TCustomDaemonApplication.GUIMainLoop	191
9.8.16	TCustomDaemonApplication.GuiHandle	191
9.8.17	TCustomDaemonApplication.RunMode	191
9.9	TCustomDaemonMapper	192
9.9.1	Description	192
9.9.2	Method overview	192
9.9.3	Property overview	192
9.9.4	TCustomDaemonMapper.Create	192
9.9.5	TCustomDaemonMapper.Destroy	192
9.9.6	TCustomDaemonMapper.DaemonDefs	193
9.9.7	TCustomDaemonMapper.OnCreate	193
9.9.8	TCustomDaemonMapper.OnDestroy	193
9.9.9	TCustomDaemonMapper.OnRun	193
9.9.10	TCustomDaemonMapper.OnInstall	194
9.9.11	TCustomDaemonMapper.OnUnInstall	194
9.10	TDaemon	194
9.10.1	Description	194
9.10.2	Property overview	195
9.10.3	TDaemon.Definition	195
9.10.4	TDaemon.Status	195
9.10.5	TDaemon.OnStart	195
9.10.6	TDaemon.OnStop	196
9.10.7	TDaemon.OnPause	196
9.10.8	TDaemon.OnContinue	196
9.10.9	TDaemon.OnShutDown	197
9.10.10	TDaemon.OnExecute	197

9.10.11	TDaemon.BeforeInstall	197
9.10.12	TDaemon.AfterInstall	198
9.10.13	TDaemon.BeforeUnInstall	198
9.10.14	TDaemon.AfterUnInstall	198
9.10.15	TDaemon.OnControlCode	198
9.11	TDaemonApplication	199
9.11.1	Description	199
9.12	TDaemonController	199
9.12.1	Description	199
9.12.2	Method overview	199
9.12.3	Property overview	199
9.12.4	TDaemonController.Create	199
9.12.5	TDaemonController.Destroy	200
9.12.6	TDaemonController.StartService	200
9.12.7	TDaemonController.Main	200
9.12.8	TDaemonController.Controller	200
9.12.9	TDaemonController.ReportStatus	201
9.12.10	TDaemonController.Daemon	201
9.12.11	TDaemonController.Params	201
9.12.12	TDaemonController.LastStatus	201
9.12.13	TDaemonController.CheckPoint	202
9.13	TDaemonDef	202
9.13.1	Description	202
9.13.2	Method overview	202
9.13.3	Property overview	202
9.13.4	TDaemonDef.Create	202
9.13.5	TDaemonDef.Destroy	203
9.13.6	TDaemonDef.DaemonClass	203
9.13.7	TDaemonDef.Instance	203
9.13.8	TDaemonDef.DaemonClassName	203
9.13.9	TDaemonDef.Name	204
9.13.10	TDaemonDef.Description	204
9.13.11	TDaemonDef.DisplayName	204
9.13.12	TDaemonDef.RunArguments	204
9.13.13	TDaemonDef.Options	205
9.13.14	TDaemonDef.Enabled	205
9.13.15	TDaemonDef.WinBindings	205
9.13.16	TDaemonDef.OnCreateInstance	205
9.13.17	TDaemonDef.LogStatusReport	206
9.14	TDaemonDefs	206

9.14.1	Description	206
9.14.2	Method overview	206
9.14.3	Property overview	206
9.14.4	TDaemonDefs.Create	206
9.14.5	TDaemonDefs.IndexOfDaemonDef	207
9.14.6	TDaemonDefs.FindDaemonDef	207
9.14.7	TDaemonDefs.DaemonDefByName	207
9.14.8	TDaemonDefs.Daemons	207
9.15	TDaemonMapper	208
9.15.1	Description	208
9.15.2	Method overview	208
9.15.3	TDaemonMapper.Create	208
9.15.4	TDaemonMapper.CreateNew	208
9.16	TDaemonThread	209
9.16.1	Description	209
9.16.2	Method overview	209
9.16.3	Property overview	209
9.16.4	TDaemonThread.Create	209
9.16.5	TDaemonThread.Execute	209
9.16.6	TDaemonThread.CheckControlMessage	210
9.16.7	TDaemonThread.StopDaemon	210
9.16.8	TDaemonThread.PauseDaemon	210
9.16.9	TDaemonThread.ContinueDaemon	210
9.16.10	TDaemonThread.ShutDownDaemon	211
9.16.11	TDaemonThread.InterrogateDaemon	211
9.16.12	TDaemonThread.Daemon	211
9.17	TDependencies	211
9.17.1	Description	211
9.17.2	Method overview	211
9.17.3	Property overview	212
9.17.4	TDependencies.Create	212
9.17.5	TDependencies.Items	212
9.18	TDependency	212
9.18.1	Description	212
9.18.2	Method overview	212
9.18.3	Property overview	212
9.18.4	TDependency.Assign	213
9.18.5	TDependency.Name	213
9.18.6	TDependency.IsGroup	213
9.19	TWinBindings	213

9.19.1	Description	213
9.19.2	Method overview	213
9.19.3	Property overview	214
9.19.4	TWinBindings.Create	214
9.19.5	TWinBindings.Destroy	214
9.19.6	TWinBindings.Assign	214
9.19.7	TWinBindings.ErrCode	214
9.19.8	TWinBindings.Win32ErrCode	215
9.19.9	TWinBindings.Dependencies	215
9.19.10	TWinBindings.GroupName	215
9.19.11	TWinBindings.Password	216
9.19.12	TWinBindings.UserName	216
9.19.13	TWinBindings.StartType	216
9.19.14	TWinBindings.WaitHint	216
9.19.15	TWinBindings.IDTag	217
9.19.16	TWinBindings.ServiceType	217
9.19.17	TWinBindings.ErrorSeverity	217
10	Reference for unit 'db'	218
10.1	Used units	218
10.2	Overview	218
10.3	Constants, types and variables	218
10.3.1	Constants	218
10.3.2	Types	219
10.4	Procedures and functions	232
10.4.1	BuffersEqual	232
10.4.2	DatabaseError	233
10.4.3	DatabaseErrorFmt	233
10.4.4	DateTimeRecToDateTime	233
10.4.5	DateTimeToDateTimeRec	234
10.4.6	DisposeMem	234
10.4.7	ExtractFieldName	234
10.4.8	SkipComments	234
10.5	EDatabaseError	235
10.5.1	Description	235
10.6	EUpdateError	235
10.6.1	Description	235
10.6.2	Method overview	235
10.6.3	Property overview	235
10.6.4	EUpdateError.Create	235

10.6.5	EUpdateError.Destroy	236
10.6.6	EUpdateError.Context	236
10.6.7	EUpdateError.ErrorCode	236
10.6.8	EUpdateError.OriginalException	236
10.6.9	EUpdateError.PreviousError	237
10.7	IProviderSupport	237
10.7.1	Method overview	237
10.7.2	IProviderSupport.PSEndTransaction	237
10.7.3	IProviderSupport.PSExecute	237
10.7.4	IProviderSupport.PSExecuteStatement	238
10.7.5	IProviderSupport.PSGetAttributes	238
10.7.6	IProviderSupport.PSGetCommandText	238
10.7.7	IProviderSupport.PSGetCommandType	238
10.7.8	IProviderSupport.PSGetDefaultOrder	238
10.7.9	IProviderSupport.PSGetIndexDefs	238
10.7.10	IProviderSupport.PSGetKeyFields	238
10.7.11	IProviderSupport.PSGetParams	238
10.7.12	IProviderSupport.PSGetQuoteChar	238
10.7.13	IProviderSupport.PSGetTableName	239
10.7.14	IProviderSupport.PSGetUpdateException	239
10.7.15	IProviderSupport.PSInTransaction	239
10.7.16	IProviderSupport.PSIsSQLBased	239
10.7.17	IProviderSupport.PSIsSQLSupported	239
10.7.18	IProviderSupport.PSReset	239
10.7.19	IProviderSupport.PSSetCommandText	239
10.7.20	IProviderSupport.PSSetParams	239
10.7.21	IProviderSupport.PSStartTransaction	239
10.7.22	IProviderSupport.PSUpdateRecord	240
10.8	TAutoIncField	240
10.8.1	Description	240
10.8.2	Method overview	240
10.8.3	TAutoIncField.Create	240
10.9	TBCDField	240
10.9.1	Description	240
10.9.2	Method overview	241
10.9.3	Property overview	241
10.9.4	TBCDField.Create	241
10.9.5	TBCDField.CheckRange	241
10.9.6	TBCDField.Value	241
10.9.7	TBCDField.Precision	242

10.9.8	TBCDField.Currency	242
10.9.9	TBCDField.MaxValue	242
10.9.10	TBCDField.MinValue	243
10.9.11	TBCDField.Size	243
10.10	TBinaryField	243
10.10.1	Description	243
10.10.2	Method overview	243
10.10.3	Property overview	243
10.10.4	TBinaryField.Create	244
10.10.5	TBinaryField.Size	244
10.11	TBlobField	244
10.11.1	Description	244
10.11.2	Method overview	244
10.11.3	Property overview	245
10.11.4	TBlobField.Create	245
10.11.5	TBlobField.Clear	245
10.11.6	TBlobField.IsBlob	245
10.11.7	TBlobField.LoadFromFile	245
10.11.8	TBlobField.LoadFromStream	246
10.11.9	TBlobField.SaveToFile	246
10.11.10	TBlobField.SaveToStream	246
10.11.11	TBlobField.SetFieldType	247
10.11.12	TBlobField.BlobSize	247
10.11.13	TBlobField.Modified	247
10.11.14	TBlobField.Value	247
10.11.15	TBlobField.Transliterate	248
10.11.16	TBlobField.BlobType	248
10.11.17	TBlobField.Size	248
10.12	TBooleanField	248
10.12.1	Description	248
10.12.2	Method overview	249
10.12.3	Property overview	249
10.12.4	TBooleanField.Create	249
10.12.5	TBooleanField.Value	249
10.12.6	TBooleanField.DisplayValues	249
10.13	TBytesField	250
10.13.1	Description	250
10.13.2	Method overview	250
10.13.3	TBytesField.Create	250
10.14	TCheckConstraint	250

10.14.1 Description	250
10.14.2 Method overview	251
10.14.3 Property overview	251
10.14.4 TCheckConstraint.Assign	251
10.14.5 TCheckConstraint.CustomConstraint	251
10.14.6 TCheckConstraint.ErrorMessage	251
10.14.7 TCheckConstraint.FromDictionary	252
10.14.8 TCheckConstraint.ImportedConstraint	252
10.15 TCheckConstraints	252
10.15.1 Description	252
10.15.2 Method overview	252
10.15.3 Property overview	252
10.15.4 TCheckConstraints.Create	253
10.15.5 TCheckConstraints.Add	253
10.15.6 TCheckConstraints.Items	253
10.16 TCurrencyField	253
10.16.1 Description	253
10.16.2 Method overview	254
10.16.3 Property overview	254
10.16.4 TCurrencyField.Create	254
10.16.5 TCurrencyField.Currency	254
10.17 TCustomConnection	254
10.17.1 Description	254
10.17.2 Method overview	254
10.17.3 Property overview	255
10.17.4 TCustomConnection.Close	255
10.17.5 TCustomConnection.Destroy	255
10.17.6 TCustomConnection.Open	255
10.17.7 TCustomConnection.DataSetCount	256
10.17.8 TCustomConnection.DataSets	256
10.17.9 TCustomConnection.Connected	256
10.17.10 TCustomConnection.LoginPrompt	257
10.17.11 TCustomConnection.AfterConnect	257
10.17.12 TCustomConnection.AfterDisconnect	257
10.17.13 TCustomConnection.BeforeConnect	257
10.17.14 TCustomConnection.BeforeDisconnect	258
10.17.15 TCustomConnection.OnLogin	258
10.18 TDatabase	258
10.18.1 Description	258
10.18.2 Method overview	259

10.18.3 Property overview	259
10.18.4 TDatabase.Create	259
10.18.5 TDatabase.Destroy	259
10.18.6 TDatabase.CloseDataSets	259
10.18.7 TDatabase.CloseTransactions	260
10.18.8 TDatabase.StartTransaction	260
10.18.9 TDatabase.EndTransaction	260
10.18.10 TDatabase.TransactionCount	260
10.18.11 TDatabase.Transactions	261
10.18.12 TDatabase.Directory	261
10.18.13 TDatabase.IsSQLBased	261
10.18.14 TDatabase.Connected	262
10.18.15 TDatabase.DatabaseName	262
10.18.16 TDatabase.KeepConnection	262
10.18.17 TDatabase.Params	262
10.19 TDataLink	263
10.19.1 Description	263
10.19.2 Method overview	263
10.19.3 Property overview	263
10.19.4 TDataLink.Create	263
10.19.5 TDataLink.Destroy	264
10.19.6 TDataLink.Edit	264
10.19.7 TDataLink.UpdateRecord	264
10.19.8 TDataLink.ExecuteAction	264
10.19.9 TDataLink.UpdateAction	265
10.19.10 TDataLink.Active	265
10.19.11 TDataLink.ActiveRecord	265
10.19.12 TDataLink.BOF	265
10.19.13 TDataLink.BufferCount	266
10.19.14 TDataLink.DataSet	266
10.19.15 TDataLink.DataSource	266
10.19.16 TDataLink.DataSourceFixed	266
10.19.17 TDataLink.Editing	267
10.19.18 TDataLink.Eof	267
10.19.19 TDataLink.ReadOnly	267
10.19.20 TDataLink.RecordCount	267
10.20 TDataSet	268
10.20.1 Description	268
10.20.2 Method overview	270
10.20.3 Property overview	271

10.20.4 TDataSet.Create	272
10.20.5 TDataSet.Destroy	272
10.20.6 TDataSet.ActiveBuffer	272
10.20.7 TDataSet.GetFieldData	272
10.20.8 TDataSet.SetFieldData	273
10.20.9 TDataSet.Append	273
10.20.10 TDataSet.AppendRecord	273
10.20.11 TDataSet.BookmarkValid	274
10.20.12 TDataSet.Cancel	274
10.20.13 TDataSet.CheckBrowseMode	274
10.20.14 TDataSet.ClearFields	274
10.20.15 TDataSet.Close	275
10.20.16 TDataSet.ControlsDisabled	275
10.20.17 TDataSet.CompareBookmarks	275
10.20.18 TDataSet.CreateBlobStream	276
10.20.19 TDataSet.CursorPosChanged	276
10.20.20 TDataSet.DataConvert	276
10.20.21 TDataSet.Delete	276
10.20.22 TDataSet.DisableControls	277
10.20.23 TDataSet.Edit	277
10.20.24 TDataSet.EnableControls	278
10.20.25 TDataSet.FieldByName	278
10.20.26 TDataSet.FindField	278
10.20.27 TDataSet.FindFirst	279
10.20.28 TDataSet.FindLast	279
10.20.29 TDataSet.FindNext	279
10.20.30 TDataSet.FindPrior	279
10.20.31 TDataSet.First	280
10.20.32 TDataSet.FreeBookmark	280
10.20.33 TDataSet.GetBookmark	280
10.20.34 TDataSet.GetCurrentRecord	281
10.20.35 TDataSet.GetFieldList	281
10.20.36 TDataSet.GetFieldNames	281
10.20.37 TDataSet.GotoBookmark	281
10.20.38 TDataSet.Insert	282
10.20.39 TDataSet.InsertRecord	282
10.20.40 TDataSet.IsEmpty	282
10.20.41 TDataSet.IsLinkedTo	282
10.20.42 TDataSet.IsSequenced	283
10.20.43 TDataSet.Last	283

10.20.44	DataSet.Locate	283
10.20.45	DataSet.Lookup	284
10.20.46	DataSet.MoveBy	284
10.20.47	DataSet.Next	284
10.20.48	DataSet.Open	285
10.20.49	DataSet.Post	285
10.20.50	DataSet.Prior	286
10.20.51	DataSet.Refresh	286
10.20.52	DataSet.Resync	286
10.20.53	DataSet.SetFields	286
10.20.54	DataSet.Translate	287
10.20.55	DataSet.UpdateCursorPos	287
10.20.56	DataSet.UpdateRecord	287
10.20.57	DataSet.UpdateStatus	288
10.20.58	DataSet.BlockReadSize	288
10.20.59	DataSet.BOF	288
10.20.60	DataSet.Bookmark	288
10.20.61	DataSet.CanModify	289
10.20.62	DataSet.DataSource	289
10.20.63	DataSet.DefaultFields	290
10.20.64	DataSet.EOF	290
10.20.65	DataSet.FieldCount	291
10.20.66	DataSet.FieldDefs	291
10.20.67	DataSet.Found	291
10.20.68	DataSet.Modified	292
10.20.69	DataSet.IsUniDirectional	292
10.20.70	DataSet.RecordCount	292
10.20.71	DataSet.RecNo	293
10.20.72	DataSet.RecordSize	293
10.20.73	DataSet.State	293
10.20.74	DataSet.Fields	294
10.20.75	DataSet.FieldValues	294
10.20.76	DataSet.Filter	294
10.20.77	DataSet.Filtered	295
10.20.78	DataSet.FilterOptions	295
10.20.79	DataSet.Active	295
10.20.80	DataSet.AutoCalcFields	296
10.20.81	DataSet.BeforeOpen	296
10.20.82	DataSet.AfterOpen	297
10.20.83	DataSet.BeforeClose	297

10.20.84	TDataset.AfterClose	297
10.20.85	TDataset.BeforeInsert	297
10.20.86	TDataset.AfterInsert	298
10.20.87	TDataset.BeforeEdit	298
10.20.88	TDataset.AfterEdit	298
10.20.89	TDataset.BeforePost	299
10.20.90	TDataset.AfterPost	299
10.20.91	TDataset.BeforeCancel	299
10.20.92	TDataset.AfterCancel	300
10.20.93	TDataset.BeforeDelete	300
10.20.94	TDataset.AfterDelete	300
10.20.95	TDataset.BeforeScroll	300
10.20.96	TDataset.AfterScroll	301
10.20.97	TDataset.BeforeRefresh	301
10.20.98	TDataset.AfterRefresh	301
10.20.99	TDataset.OnCalcFields	302
10.20.100	TDataset.OnDeleteError	302
10.20.101	TDataset.OnEditError	302
10.20.102	TDataset.OnFilterRecord	303
10.20.103	TDataset.OnNewRecord	303
10.20.104	TDataset.OnPostError	304
10.21	TDataSource	304
10.21.1	Description	304
10.21.2	Method overview	304
10.21.3	Property overview	305
10.21.4	TDataSource.Create	305
10.21.5	TDataSource.Destroy	305
10.21.6	TDataSource.Edit	305
10.21.7	TDataSource.IsLinkedTo	306
10.21.8	TDataSource.State	306
10.21.9	TDataSource.AutoEdit	306
10.21.10	TDataSource.DataSet	306
10.21.11	TDataSource.Enabled	307
10.21.12	TDataSource.OnStateChange	307
10.21.13	TDataSource.OnDataChange	307
10.21.14	TDataSource.OnUpdateData	308
10.22	TDateField	308
10.22.1	Description	308
10.22.2	Method overview	308
10.22.3	TDateField.Create	308

10.23TDateTimeField	308
10.23.1 Description	308
10.23.2 Method overview	309
10.23.3 Property overview	309
10.23.4 TDateTimeField.Create	309
10.23.5 TDateTimeField.Value	309
10.23.6 TDateTimeField.DisplayFormat	309
10.23.7 TDateTimeField.EditMask	310
10.24TDBDataset	310
10.24.1 Description	310
10.24.2 Method overview	310
10.24.3 Property overview	310
10.24.4 TDBDataset.destroy	311
10.24.5 TDBDataset.DataBase	311
10.24.6 TDBDataset.Transaction	311
10.25TDBTransaction	311
10.25.1 Description	311
10.25.2 Method overview	312
10.25.3 Property overview	312
10.25.4 TDBTransaction.Create	312
10.25.5 TDBTransaction.destroy	312
10.25.6 TDBTransaction.CloseDataSets	312
10.25.7 TDBTransaction.DataBase	313
10.25.8 TDBTransaction.Active	313
10.26TDefCollection	313
10.26.1 Description	313
10.26.2 Method overview	313
10.26.3 Property overview	313
10.26.4 TDefCollection.create	314
10.26.5 TDefCollection.Find	314
10.26.6 TDefCollection.GetItemNames	314
10.26.7 TDefCollection.IndexOf	314
10.26.8 TDefCollection.Dataset	315
10.26.9 TDefCollection.Updated	315
10.27TDetailDataLink	315
10.27.1 Description	315
10.27.2 Property overview	315
10.27.3 TDetailDataLink.DetailDataSet	315
10.28TField	316
10.28.1 Description	316

10.28.2 Method overview	316
10.28.3 Property overview	318
10.28.4 TField.Create	319
10.28.5 TField.Destroy	319
10.28.6 TField.Assign	319
10.28.7 TField.AssignValue	319
10.28.8 TField.Clear	320
10.28.9 TField.FocusControl	320
10.28.10 TField.GetData	320
10.28.11 TField.IsBlob	321
10.28.12 TField.IsValidChar	321
10.28.13 TField.RefreshLookupList	321
10.28.14 TField.SetData	321
10.28.15 TField.SetFieldType	322
10.28.16 TField.Validate	322
10.28.17 TField.AsBCD	322
10.28.18 TField.AsBoolean	323
10.28.19 TField.AsBytes	323
10.28.20 TField.AsCurrency	323
10.28.21 TField.AsDateTime	323
10.28.22 TField.AsFloat	324
10.28.23 TField.AsLongint	324
10.28.24 TField.AsLargeInt	324
10.28.25 TField.AsInteger	325
10.28.26 TField.AsString	325
10.28.27 TField.AsWideString	325
10.28.28 TField.AsVariant	326
10.28.29 TField.AttributeSet	326
10.28.30 TField.Calculated	326
10.28.31 TField.CanModify	327
10.28.32 TField.CurValue	327
10.28.33 TField.DataSet	327
10.28.34 TField.DataSize	327
10.28.35 TField.DataType	328
10.28.36 TField.DisplayName	328
10.28.37 TField.DisplayText	328
10.28.38 TField.EditMask	328
10.28.39 TField.EditMaskPtr	329
10.28.40 TField.FieldNo	329
10.28.41 TField.IsIndexField	329

10.28.42	Field.IsNull	330
10.28.43	Field.Lookup	330
10.28.44	Field.NewValue	330
10.28.45	Field.Offset	330
10.28.46	Field.Size	331
10.28.47	Field.Text	331
10.28.48	Field.ValidChars	331
10.28.49	Field.Value	332
10.28.50	Field.OldValue	332
10.28.51	Field.LookupList	332
10.28.52	Field.Alignment	333
10.28.53	Field.CustomConstraint	333
10.28.54	Field.ConstraintErrorMessage	333
10.28.55	Field.DefaultExpression	334
10.28.56	Field.DisplayLabel	334
10.28.57	Field.DisplayWidth	334
10.28.58	Field.FieldKind	334
10.28.59	Field.FieldName	335
10.28.60	Field.HasConstraints	335
10.28.61	Field.Index	335
10.28.62	Field.ImportedConstraint	335
10.28.63	Field.KeyFields	336
10.28.64	Field.LookupCache	336
10.28.65	Field.LookupDataSet	336
10.28.66	Field.LookupKeyFields	337
10.28.67	Field.LookupResultField	337
10.28.68	Field.Origin	337
10.28.69	Field.ProviderFlags	337
10.28.70	Field.ReadOnly	338
10.28.71	Field.Required	338
10.28.72	Field.Visible	338
10.28.73	Field.OnChange	339
10.28.74	Field.OnGetText	339
10.28.75	Field.OnSetText	339
10.28.76	Field.OnValidate	339
10.29	TFieldDef	340
10.29.1	Description	340
10.29.2	Method overview	340
10.29.3	Property overview	340
10.29.4	TFieldDef.Create	340

10.29.5 TFieldDef.Destroy	341
10.29.6 TFieldDef.Assign	341
10.29.7 TFieldDef.CreateField	341
10.29.8 TFieldDef.FieldClass	342
10.29.9 TFieldDef.FieldNo	342
10.29.10 TFieldDef.InternalCalcField	342
10.29.11 TFieldDef.Required	342
10.29.12 TFieldDef.Attributes	343
10.29.13 TFieldDef.DataType	343
10.29.14 TFieldDef.Precision	343
10.29.15 TFieldDef.Size	343
10.30 TFieldDefs	344
10.30.1 Description	344
10.30.2 Method overview	344
10.30.3 Property overview	344
10.30.4 TFieldDefs.Create	344
10.30.5 TFieldDefs.Add	344
10.30.6 TFieldDefs.AddFieldDef	345
10.30.7 TFieldDefs.Assign	345
10.30.8 TFieldDefs.Find	345
10.30.9 TFieldDefs.Update	345
10.30.10 TFieldDefs.MakeNameUnique	346
10.30.11 TFieldDefs.HiddenFields	346
10.30.12 TFieldDefs.Items	346
10.31 Tfields	346
10.31.1 Description	346
10.31.2 Method overview	347
10.31.3 Property overview	347
10.31.4 Tfields.Create	347
10.31.5 Tfields.Destroy	347
10.31.6 Tfields.Add	347
10.31.7 Tfields.CheckFieldName	348
10.31.8 Tfields.CheckFieldNames	348
10.31.9 Tfields.Clear	348
10.31.10 Tfields.FindField	349
10.31.11 Tfields.FieldByName	349
10.31.12 Tfields.FieldByNumber	349
10.31.13 Tfields.GetEnumerator	349
10.31.14 Tfields.GetFieldNames	350
10.31.15 Tfields.IndexOf	350

10.31.16	fields.Remove	350
10.31.17	fields.Count	350
10.31.18	fields.Dataset	351
10.31.19	fields.Fields	351
10.32	TFieldsEnumerator	351
10.32.1	Description	351
10.32.2	Method overview	352
10.32.3	Property overview	352
10.32.4	TFieldsEnumerator.Create	352
10.32.5	TFieldsEnumerator.MoveNext	352
10.32.6	TFieldsEnumerator.Current	352
10.33	TFloatField	353
10.33.1	Description	353
10.33.2	Method overview	353
10.33.3	Property overview	353
10.33.4	TFloatField.Create	353
10.33.5	TFloatField.CheckRange	353
10.33.6	TFloatField.Value	354
10.33.7	TFloatField.Currency	354
10.33.8	TFloatField.MaxValue	354
10.33.9	TFloatField.MinValue	355
10.33.10	TFloatField.Precision	355
10.34	TFMTBCDField	355
10.34.1	Method overview	355
10.34.2	Property overview	355
10.34.3	TFMTBCDField.Create	355
10.34.4	TFMTBCDField.CheckRange	356
10.34.5	TFMTBCDField.Value	356
10.34.6	TFMTBCDField.Precision	356
10.34.7	TFMTBCDField.Currency	356
10.34.8	TFMTBCDField.MaxValue	356
10.34.9	TFMTBCDField.MinValue	356
10.34.10	TFMTBCDField.Size	356
10.35	TGraphicField	357
10.35.1	Description	357
10.35.2	Method overview	357
10.35.3	TGraphicField.Create	357
10.36	TGuidField	357
10.36.1	Description	357
10.36.2	Method overview	357

10.36.3 Property overview	358
10.36.4 TGuidField.Create	358
10.36.5 TGuidField.AsGuid	358
10.37 TIndexDef	358
10.37.1 Description	358
10.37.2 Method overview	358
10.37.3 Property overview	358
10.37.4 TIndexDef.Create	359
10.37.5 TIndexDef.Expression	359
10.37.6 TIndexDef.Fields	359
10.37.7 TIndexDef.CaseInsFields	360
10.37.8 TIndexDef.DescFields	360
10.37.9 TIndexDef.Options	360
10.37.10 TIndexDef.Source	360
10.38 TIndexDefs	361
10.38.1 Description	361
10.38.2 Method overview	361
10.38.3 Property overview	361
10.38.4 TIndexDefs.Create	361
10.38.5 TIndexDefs.Add	361
10.38.6 TIndexDefs.AddIndexDef	362
10.38.7 TIndexDefs.Find	362
10.38.8 TIndexDefs.FindIndexForFields	362
10.38.9 TIndexDefs.GetIndexForFields	362
10.38.10 TIndexDefs.Update	363
10.38.11 TIndexDefs.Items	363
10.39 TIntegerField	363
10.39.1 Description	363
10.40 TLargeintField	363
10.40.1 Description	363
10.40.2 Method overview	363
10.40.3 Property overview	364
10.40.4 TLargeintField.Create	364
10.40.5 TLargeintField.CheckRange	364
10.40.6 TLargeintField.Value	364
10.40.7 TLargeintField.MaxValue	364
10.40.8 TLargeintField.MinValue	365
10.41 TLongintField	365
10.41.1 Description	365
10.41.2 Method overview	365

10.41.3 Property overview	365
10.41.4 TLongintField.Create	366
10.41.5 TLongintField.CheckRange	366
10.41.6 TLongintField.Value	366
10.41.7 TLongintField.MaxValue	366
10.41.8 TLongintField.MinValue	367
10.42 TLookupList	367
10.42.1 Description	367
10.42.2 Method overview	367
10.42.3 TLookupList.Create	367
10.42.4 TLookupList.Destroy	367
10.42.5 TLookupList.Add	368
10.42.6 TLookupList.Clear	368
10.42.7 TLookupList.FirstKeyByValue	368
10.42.8 TLookupList.ValueOfKey	368
10.42.9 TLookupList.ValuesToStrings	369
10.43 TMasterDataLink	369
10.43.1 Description	369
10.43.2 Method overview	369
10.43.3 Property overview	369
10.43.4 TMasterDataLink.Create	369
10.43.5 TMasterDataLink.Destroy	370
10.43.6 TMasterDataLink.FieldNames	370
10.43.7 TMasterDataLink.Fields	370
10.43.8 TMasterDataLink.OnMasterChange	370
10.43.9 TMasterDataLink.OnMasterDisable	371
10.44 TMasterParamsDataLink	371
10.44.1 Description	371
10.44.2 Method overview	371
10.44.3 Property overview	371
10.44.4 TMasterParamsDataLink.Create	371
10.44.5 TMasterParamsDataLink.RefreshParamNames	372
10.44.6 TMasterParamsDataLink.CopyParamsFromMaster	372
10.44.7 TMasterParamsDataLink.Params	372
10.45 TMemoField	372
10.45.1 Description	372
10.45.2 Method overview	373
10.45.3 Property overview	373
10.45.4 TMemoField.Create	373
10.45.5 TMemoField.Transliterate	373

10.46	TNamedItem	373
10.46.1	Description	373
10.46.2	Property overview	373
10.46.3	TNamedItem.DisplayName	374
10.46.4	TNamedItem.Name	374
10.47	TNumericField	374
10.47.1	Description	374
10.47.2	Method overview	374
10.47.3	Property overview	374
10.47.4	TNumericField.Create	375
10.47.5	TNumericField.Alignment	375
10.47.6	TNumericField.DisplayFormat	375
10.47.7	TNumericField.EditFormat	375
10.48	TParam	376
10.48.1	Description	376
10.48.2	Method overview	376
10.48.3	Property overview	377
10.48.4	TParam.Create	377
10.48.5	TParam.Assign	377
10.48.6	TParam.AssignField	378
10.48.7	TParam.AssignToField	378
10.48.8	TParam.AssignFieldValue	378
10.48.9	TParam.AssignFromField	379
10.48.10	TParam.Clear	379
10.48.11	TParam.GetData	379
10.48.12	TParam.GetDataSize	379
10.48.13	TParam.LoadFromFile	380
10.48.14	TParam.LoadFromStream	380
10.48.15	TParam.SetBlobData	380
10.48.16	TParam.SetData	380
10.48.17	TParam.AsBlob	381
10.48.18	TParam.AsBoolean	381
10.48.19	TParam.AsCurrency	381
10.48.20	TParam.AsDate	382
10.48.21	TParam.AsDateTime	382
10.48.22	TParam.AsFloat	382
10.48.23	TParam.AsInteger	382
10.48.24	TParam.AsLargeInt	383
10.48.25	TParam.AsMemo	383
10.48.26	TParam.AsSmallInt	383

10.48.27	TPParam.AsString	383
10.48.28	TPParam.AsTime	384
10.48.29	TPParam.AsWord	384
10.48.30	TPParam.AsFMTBCD	384
10.48.31	TPParam.Bound	384
10.48.32	TPParam.Dataset	385
10.48.33	TPParam.IsNull	385
10.48.34	TPParam.NativeStr	385
10.48.35	TPParam.Text	385
10.48.36	TPParam.Value	386
10.48.37	TPParam.AsWideString	386
10.48.38	TPParam.DataType	386
10.48.39	TPParam.Name	386
10.48.40	TPParam.NumericScale	387
10.48.41	TPParam.ParamType	387
10.48.42	TPParam.Precision	387
10.48.43	TPParam.Size	388
10.49	TParams	388
10.49.1	Description	388
10.49.2	Method overview	388
10.49.3	Property overview	389
10.49.4	TParams.Create	389
10.49.5	TParams.AddParam	389
10.49.6	TParams.AssignValues	389
10.49.7	TParams.CreateParam	389
10.49.8	TParams.FindParam	390
10.49.9	TParams.GetParamList	390
10.49.10	TParams.IsEqual	390
10.49.11	TParams.ParamByName	391
10.49.12	TParams.ParseSQL	391
10.49.13	TParams.RemoveParam	392
10.49.14	TParams.CopyParamValuesFromDataset	392
10.49.15	TParams.Dataset	392
10.49.16	TParams.Items	393
10.49.17	TParams.ParamValues	393
10.50	TSmallintField	393
10.50.1	Description	393
10.50.2	Method overview	393
10.50.3	TSmallintField.Create	394
10.51	TStringField	394

10.51.1 Description	394
10.51.2 Method overview	394
10.51.3 Property overview	394
10.51.4 TStringField.Create	394
10.51.5 TStringField.SetFieldType	395
10.51.6 TStringField.FixedChar	395
10.51.7 TStringField.Transliterate	395
10.51.8 TStringField.Value	395
10.51.9 TStringField.EditMask	396
10.51.10 TStringField.Size	396
10.52 TTimeField	396
10.52.1 Description	396
10.52.2 Method overview	396
10.52.3 TTimeField.Create	397
10.53 TVarBytesField	397
10.53.1 Description	397
10.53.2 Method overview	397
10.53.3 TVarBytesField.Create	397
10.54 TVariantField	397
10.54.1 Description	397
10.54.2 Method overview	398
10.54.3 TVariantField.Create	398
10.55 TWideMemoField	398
10.55.1 Description	398
10.55.2 Method overview	398
10.55.3 Property overview	398
10.55.4 TWideMemoField.Create	398
10.55.5 TWideMemoField.Value	399
10.56 TWideStringField	399
10.56.1 Description	399
10.56.2 Method overview	399
10.56.3 Property overview	399
10.56.4 TWideStringField.Create	399
10.56.5 TWideStringField.SetFieldType	400
10.56.6 TWideStringField.Value	400
10.57 TWordField	400
10.57.1 Description	400
10.57.2 Method overview	400
10.57.3 TWordField.Create	400

11 Reference for unit 'dbugintf'	401
11.1 Overview	401
11.2 Writing a debug server	401
11.3 Constants, types and variables	401
11.3.1 Resource strings	401
11.3.2 Constants	402
11.3.3 Types	402
11.4 Procedures and functions	402
11.4.1 GetDebuggingEnabled	402
11.4.2 InitDebugClient	403
11.4.3 SendBoolean	403
11.4.4 SendDateTime	403
11.4.5 SendDebug	403
11.4.6 SendDebugEx	404
11.4.7 SendDebugFmt	404
11.4.8 SendDebugFmtEx	404
11.4.9 SendInteger	405
11.4.10 SendMethodEnter	405
11.4.11 SendMethodExit	405
11.4.12 SendPointer	406
11.4.13 SendSeparator	406
11.4.14 SetDebuggingEnabled	406
11.4.15 StartDebugServer	406
12 Reference for unit 'dbugmsg'	408
12.1 Used units	408
12.2 Overview	408
12.3 Constants, types and variables	408
12.3.1 Constants	408
12.3.2 Types	409
12.4 Procedures and functions	409
12.4.1 DebugMessageName	409
12.4.2 ReadDebugMessageFromStream	409
12.4.3 WriteDebugMessageToStream	410
13 Reference for unit 'eventlog'	411
13.1 Used units	411
13.2 Overview	411
13.3 Constants, types and variables	411
13.3.1 Resource strings	411
13.3.2 Types	412

13.4	ELogError	412
13.4.1	Description	412
13.5	TEventLog	413
13.5.1	Description	413
13.5.2	Method overview	413
13.5.3	Property overview	413
13.5.4	TEventLog.Destroy	413
13.5.5	TEventLog.EventTypeToString	414
13.5.6	TEventLog.RegisterMessageFile	414
13.5.7	TEventLog.UnRegisterMessageFile	415
13.5.8	TEventLog.Pause	415
13.5.9	TEventLog.Resume	415
13.5.10	TEventLog.Log	415
13.5.11	TEventLog.Warning	416
13.5.12	TEventLog.Error	416
13.5.13	TEventLog.Debug	416
13.5.14	TEventLog.Info	416
13.5.15	TEventLog.AppendContent	417
13.5.16	TEventLog.Identification	417
13.5.17	TEventLog.LogType	417
13.5.18	TEventLog.Active	417
13.5.19	TEventLog.RaiseExceptionOnError	418
13.5.20	TEventLog.DefaultEventType	418
13.5.21	TEventLog.FileName	418
13.5.22	TEventLog.TimeStampFormat	419
13.5.23	TEventLog.CustomLogType	419
13.5.24	TEventLog.EventIDOffset	419
13.5.25	TEventLog.OnGetCustomCategory	420
13.5.26	TEventLog.OnGetCustomEventID	420
13.5.27	TEventLog.OnGetCustomEvent	420
13.5.28	TEventLog.Paused	420
14	Reference for unit 'ezcgi'	421
14.1	Used units	421
14.2	Overview	421
14.3	Constants, types and variables	421
14.3.1	Constants	421
14.4	ECGIException	421
14.4.1	Description	421
14.5	TEZcgi	422

14.5.1	Description	422
14.5.2	Method overview	422
14.5.3	Property overview	422
14.5.4	TEZcgi.Create	422
14.5.5	TEZcgi.Destroy	422
14.5.6	TEZcgi.Run	423
14.5.7	TEZcgi.WriteContent	423
14.5.8	TEZcgi.PutLine	423
14.5.9	TEZcgi.GetValue	424
14.5.10	TEZcgi.DoPost	424
14.5.11	TEZcgi.DoGet	424
14.5.12	TEZcgi.Values	424
14.5.13	TEZcgi.Names	425
14.5.14	TEZcgi.Variables	425
14.5.15	TEZcgi.VariableCount	426
14.5.16	TEZcgi.Name	426
14.5.17	TEZcgi.Email	426
15	Reference for unit 'fpTimer'	427
15.1	Used units	427
15.2	Overview	427
15.3	Constants, types and variables	427
15.3.1	Types	427
15.3.2	Variables	427
15.4	TFPCustomTimer	428
15.4.1	Description	428
15.4.2	Method overview	428
15.4.3	TFPCustomTimer.Create	428
15.4.4	TFPCustomTimer.Destroy	428
15.4.5	TFPCustomTimer.StartTimer	429
15.4.6	TFPCustomTimer.StopTimer	429
15.5	TFPTimer	429
15.5.1	Description	429
15.5.2	Property overview	429
15.5.3	TFPTimer.Enabled	429
15.5.4	TFPTimer.Interval	430
15.5.5	TFPTimer.OnTimer	430
15.6	TFPTimerDriver	430
15.6.1	Description	430
15.6.2	Method overview	430

15.6.3	Property overview	430
15.6.4	TFPTimerDriver.Create	431
15.6.5	TFPTimerDriver.StartTimer	431
15.6.6	TFPTimerDriver.StopTimer	431
15.6.7	TFPTimerDriver.Timer	431
16	Reference for unit 'gettext'	432
16.1	Used units	432
16.2	Overview	432
16.3	Constants, types and variables	432
16.3.1	Constants	432
16.3.2	Types	432
16.4	Procedures and functions	433
16.4.1	GetLanguageIDs	433
16.4.2	TranslateResourceStrings	434
16.4.3	TranslateUnitResourceStrings	434
16.5	EMOFileError	434
16.5.1	Description	434
16.6	TMOFile	434
16.6.1	Description	434
16.6.2	Method overview	435
16.6.3	TMOFile.Create	435
16.6.4	TMOFile.Destroy	435
16.6.5	TMOFile.Translate	435
17	Reference for unit 'IBConnection'	436
17.1	Used units	436
17.2	Constants, types and variables	436
17.2.1	Constants	436
17.3	EIBDatabaseError	436
17.3.1	Description	436
17.4	TIBConnection	437
17.4.1	Description	437
17.4.2	Method overview	437
17.4.3	Property overview	438
17.4.4	TIBConnection.Create	438
17.4.5	TIBConnection.CreateDB	438
17.4.6	TIBConnection.DropDB	438
17.4.7	TIBConnection.GetDBDialect	439
17.4.8	TIBConnection.BlobSegmentSize	439
17.4.9	TIBConnection.DatabaseName	439

17.4.10	TIBConnection.Dialect	440
17.4.11	TIBConnection.KeepConnection	440
17.4.12	TIBConnection.LoginPrompt	440
17.4.13	TIBConnection.Params	441
17.4.14	TIBConnection.OnLogin	441
17.5	TIBConnectionDef	441
17.5.1	Description	441
17.5.2	Method overview	441
17.5.3	TIBConnectionDef.TypeName	442
17.5.4	TIBConnectionDef.ConnectionClass	442
17.5.5	TIBConnectionDef.Description	442
17.5.6	TIBConnectionDef.DefaultLibraryName	442
17.5.7	TIBConnectionDef.LoadFunction	442
17.5.8	TIBConnectionDef.UnLoadFunction	442
17.6	TIBCursor	443
17.6.1	Description	443
17.7	TIBTrans	443
17.7.1	Description	443
18	Reference for unit 'idea'	444
18.1	Used units	444
18.2	Overview	444
18.3	Constants, types and variables	444
18.3.1	Constants	444
18.3.2	Types	445
18.4	Procedures and functions	445
18.4.1	CipherIdea	445
18.4.2	DeKeyIdea	445
18.4.3	EnKeyIdea	446
18.5	EIDEAError	446
18.5.1	Description	446
18.6	TIDEADeCryptStream	446
18.6.1	Description	446
18.6.2	Method overview	446
18.6.3	TIDEADeCryptStream.Create	447
18.6.4	TIDEADeCryptStream.Read	447
18.6.5	TIDEADeCryptStream.Seek	447
18.7	TIDEAEncryptStream	448
18.7.1	Description	448
18.7.2	Method overview	448

18.7.3	TIDEAEncryptStream.Create	448
18.7.4	TIDEAEncryptStream.Destroy	448
18.7.5	TIDEAEncryptStream.Write	449
18.7.6	TIDEAEncryptStream.Seek	449
18.7.7	TIDEAEncryptStream.Flush	449
18.8	TIDEAStream	449
18.8.1	Description	449
18.8.2	Method overview	450
18.8.3	Property overview	450
18.8.4	TIDEAStream.Create	450
18.8.5	TIDEAStream.Key	450
19	Reference for unit 'inicol'	451
19.1	Used units	451
19.2	Overview	451
19.3	Constants, types and variables	451
19.3.1	Constants	451
19.4	EIniCol	452
19.4.1	Description	452
19.5	TIniCollection	452
19.5.1	Description	452
19.5.2	Method overview	452
19.5.3	Property overview	452
19.5.4	TIniCollection.Load	452
19.5.5	TIniCollection.Save	453
19.5.6	TIniCollection.SaveToIni	453
19.5.7	TIniCollection.SaveToFile	453
19.5.8	TIniCollection.LoadFromIni	454
19.5.9	TIniCollection.LoadFromFile	454
19.5.10	TIniCollection.Prefix	454
19.5.11	TIniCollection.SectionPrefix	455
19.5.12	TIniCollection.FileName	455
19.5.13	TIniCollection.GlobalSection	455
19.6	TIniCollectionItem	456
19.6.1	Description	456
19.6.2	Method overview	456
19.6.3	Property overview	456
19.6.4	TIniCollectionItem.SaveToIni	456
19.6.5	TIniCollectionItem.LoadFromIni	456
19.6.6	TIniCollectionItem.SaveToFile	457

19.6.7	TIniCollectionItem.LoadFromFile	457
19.6.8	TIniCollectionItem.SectionName	457
19.7	TNamedIniCollection	458
19.7.1	Description	458
19.7.2	Method overview	458
19.7.3	Property overview	458
19.7.4	TNamedIniCollection.IndexOfUserData	458
19.7.5	TNamedIniCollection.IndexOfName	458
19.7.6	TNamedIniCollection.FindByName	459
19.7.7	TNamedIniCollection.FindByUserData	459
19.7.8	TNamedIniCollection.NamedItems	459
19.8	TNamedIniCollectionItem	459
19.8.1	Description	459
19.8.2	Property overview	459
19.8.3	TNamedIniCollectionItem.UserData	460
19.8.4	TNamedIniCollectionItem.Name	460
20	Reference for unit 'IniFiles'	461
20.1	Used units	461
20.2	Overview	461
20.3	TCustomIniFile	461
20.3.1	Description	461
20.3.2	Method overview	462
20.3.3	Property overview	462
20.3.4	TCustomIniFile.Create	462
20.3.5	TCustomIniFile.Destroy	463
20.3.6	TCustomIniFile.SectionExists	463
20.3.7	TCustomIniFile.ReadString	463
20.3.8	TCustomIniFile.WriteString	464
20.3.9	TCustomIniFile.ReadInteger	464
20.3.10	TCustomIniFile.WriteInteger	464
20.3.11	TCustomIniFile.ReadBool	464
20.3.12	TCustomIniFile.WriteBool	465
20.3.13	TCustomIniFile.ReadDate	465
20.3.14	TCustomIniFile.ReadDateTime	465
20.3.15	TCustomIniFile.ReadFloat	466
20.3.16	TCustomIniFile.ReadTime	466
20.3.17	TCustomIniFile.ReadBinaryStream	466
20.3.18	TCustomIniFile.WriteDate	467
20.3.19	TCustomIniFile.WriteDateTime	467

20.3.20	TCustomIniFile.WriteFloat	467
20.3.21	TCustomIniFile.WriteTime	467
20.3.22	TCustomIniFile.WriteBinaryStream	468
20.3.23	TCustomIniFile.ReadSection	468
20.3.24	TCustomIniFile.ReadSections	468
20.3.25	TCustomIniFile.ReadSectionValues	469
20.3.26	TCustomIniFile.EraseSection	469
20.3.27	TCustomIniFile.DeleteKey	469
20.3.28	TCustomIniFile.UpdateFile	469
20.3.29	TCustomIniFile.ValueExists	470
20.3.30	TCustomIniFile.FileName	470
20.3.31	TCustomIniFile.EscapeLineFeeds	470
20.3.32	TCustomIniFile.CaseSensitive	470
20.3.33	TCustomIniFile.StripQuotes	471
20.4	THashedStringList	471
20.4.1	Description	471
20.4.2	Method overview	471
20.4.3	THashedStringList.Destroy	471
20.4.4	THashedStringList.IndexOf	471
20.4.5	THashedStringList.IndexOfName	472
20.5	TIniFile	472
20.5.1	Description	472
20.5.2	Method overview	472
20.5.3	Property overview	472
20.5.4	TIniFile.Create	472
20.5.5	TIniFile.Destroy	473
20.5.6	TIniFile.ReadString	473
20.5.7	TIniFile.WriteString	473
20.5.8	TIniFile.ReadSection	474
20.5.9	TIniFile.ReadSectionRaw	474
20.5.10	TIniFile.ReadSections	474
20.5.11	TIniFile.ReadSectionValues	474
20.5.12	TIniFile.EraseSection	475
20.5.13	TIniFile.DeleteKey	475
20.5.14	TIniFile.UpdateFile	475
20.5.15	TIniFile.Stream	475
20.5.16	TIniFile.CacheUpdates	476
20.6	TIniFileKey	476
20.6.1	Description	476
20.6.2	Method overview	476

20.6.3	Property overview	476
20.6.4	TIniFileKey.Create	476
20.6.5	TIniFileKey.Ident	477
20.6.6	TIniFileKey.Value	477
20.7	TIniFileKeyList	477
20.7.1	Description	477
20.7.2	Method overview	477
20.7.3	Property overview	477
20.7.4	TIniFileKeyList.Destroy	477
20.7.5	TIniFileKeyList.Clear	478
20.7.6	TIniFileKeyList.Items	478
20.8	TIniFileSection	478
20.8.1	Description	478
20.8.2	Method overview	478
20.8.3	Property overview	478
20.8.4	TIniFileSection.Empty	479
20.8.5	TIniFileSection.Create	479
20.8.6	TIniFileSection.Destroy	479
20.8.7	TIniFileSection.Name	479
20.8.8	TIniFileSection.KeyList	480
20.9	TIniFileSectionList	480
20.9.1	Description	480
20.9.2	Method overview	480
20.9.3	Property overview	480
20.9.4	TIniFileSectionList.Destroy	480
20.9.5	TIniFileSectionList.Clear	480
20.9.6	TIniFileSectionList.Items	481
20.10	TMemIniFile	481
20.10.1	Description	481
20.10.2	Method overview	481
20.10.3	TMemIniFile.Create	481
20.10.4	TMemIniFile.Clear	482
20.10.5	TMemIniFile.GetStrings	482
20.10.6	TMemIniFile.Rename	482
20.10.7	TMemIniFile.SetStrings	482
21	Reference for unit 'iostream'	483
21.1	Used units	483
21.2	Overview	483
21.3	Constants, types and variables	483

21.3.1	Types	483
21.4	EIOStreamError	484
21.4.1	Description	484
21.5	TIOStream	484
21.5.1	Description	484
21.5.2	Method overview	484
21.5.3	TIOStream.Create	484
21.5.4	TIOStream.Read	484
21.5.5	TIOStream.Write	485
21.5.6	TIOStream.Seek	485
22	Reference for unit 'libtar'	486
22.1	Used units	486
22.2	Overview	486
22.3	Constants, types and variables	486
22.3.1	Constants	486
22.3.2	Types	487
22.4	Procedures and functions	489
22.4.1	ClearDirRec	489
22.4.2	ConvertFilename	489
22.4.3	FileTimeGMT	489
22.4.4	PermissionString	489
22.5	TTarArchive	490
22.5.1	Description	490
22.5.2	Method overview	490
22.5.3	TTarArchive.Create	490
22.5.4	TTarArchive.Destroy	490
22.5.5	TTarArchive.Reset	490
22.5.6	TTarArchive.FindNext	491
22.5.7	TTarArchive.ReadFile	491
22.5.8	TTarArchive.GetFilePos	491
22.5.9	TTarArchive.SetFilePos	492
22.6	TTarWriter	492
22.6.1	Description	492
22.6.2	Method overview	492
22.6.3	Property overview	492
22.6.4	TTarWriter.Create	492
22.6.5	TTarWriter.Destroy	493
22.6.6	TTarWriter.AddFile	493
22.6.7	TTarWriter.AddStream	493

22.6.8	TTarWriter.AddString	494
22.6.9	TTarWriter.AddDir	494
22.6.10	TTarWriter.AddSymbolicLink	494
22.6.11	TTarWriter.AddLink	495
22.6.12	TTarWriter.AddVolumeHeader	495
22.6.13	TTarWriter.Finalize	495
22.6.14	TTarWriter.Permissions	495
22.6.15	TTarWriter.UID	496
22.6.16	TTarWriter.GID	496
22.6.17	TTarWriter.UserName	496
22.6.18	TTarWriter.GroupName	496
22.6.19	TTarWriter.Mode	497
22.6.20	TTarWriter.Magic	497
23	Reference for unit 'mssqlconn'	498
23.1	Used units	498
23.2	Overview	498
23.3	Constants, types and variables	498
23.3.1	Types	498
23.3.2	Variables	499
23.4	EMSSQLDatabaseError	499
23.4.1	Description	499
23.5	TMSSQLConnection	499
23.5.1	Description	499
23.5.2	Method overview	499
23.5.3	Property overview	500
23.5.4	TMSSQLConnection.Create	500
23.5.5	TMSSQLConnection.Password	500
23.5.6	TMSSQLConnection.Transaction	500
23.5.7	TMSSQLConnection.UserName	500
23.5.8	TMSSQLConnection.CharSet	501
23.5.9	TMSSQLConnection.HostName	501
23.5.10	TMSSQLConnection.Connected	501
23.5.11	TMSSQLConnection.Role	501
23.5.12	TMSSQLConnection.DatabaseName	501
23.5.13	TMSSQLConnection.KeepConnection	502
23.5.14	TMSSQLConnection.LoginPrompt	502
23.5.15	TMSSQLConnection.Params	502
23.5.16	TMSSQLConnection.OnLogin	502
23.6	TMSSQLConnectionDef	502

23.6.1	Method overview	502
23.6.2	TMSSQLConnectionDef.TypeName	502
23.6.3	TMSSQLConnectionDef.ConnectionClass	503
23.6.4	TMSSQLConnectionDef.Description	503
23.7	TSybaseConnection	503
23.7.1	Description	503
23.7.2	Method overview	503
23.7.3	TSybaseConnection.Create	503
23.8	TSybaseConnectionDef	503
23.8.1	Method overview	503
23.8.2	TSybaseConnectionDef.TypeName	504
23.8.3	TSybaseConnectionDef.ConnectionClass	504
23.8.4	TSybaseConnectionDef.Description	504
24	Reference for unit 'Pipes'	505
24.1	Used units	505
24.2	Overview	505
24.3	Constants, types and variables	505
24.3.1	Constants	505
24.4	Procedures and functions	505
24.4.1	CreatePipeHandles	505
24.4.2	CreatePipeStreams	506
24.5	EPipeCreation	506
24.5.1	Description	506
24.6	EPipeError	506
24.6.1	Description	506
24.7	EPipeSeek	506
24.7.1	Description	506
24.8	TInputPipeStream	506
24.8.1	Description	506
24.8.2	Method overview	507
24.8.3	Property overview	507
24.8.4	TInputPipeStream.Destroy	507
24.8.5	TInputPipeStream.Write	507
24.8.6	TInputPipeStream.Seek	507
24.8.7	TInputPipeStream.Read	508
24.8.8	TInputPipeStream.NumBytesAvailable	508
24.9	TOutputPipeStream	508
24.9.1	Description	508
24.9.2	Method overview	508

24.9.3	TOutputPipeStream.Destroy	509
24.9.4	TOutputPipeStream.Seek	509
24.9.5	TOutputPipeStream.Read	509
25	Reference for unit 'pooledmm'	510
25.1	Used units	510
25.2	Overview	510
25.3	Constants, types and variables	510
25.3.1	Types	510
25.4	TNonFreePooledMemManager	511
25.4.1	Description	511
25.4.2	Method overview	511
25.4.3	Property overview	511
25.4.4	TNonFreePooledMemManager.Clear	511
25.4.5	TNonFreePooledMemManager.Create	511
25.4.6	TNonFreePooledMemManager.Destroy	512
25.4.7	TNonFreePooledMemManager.NewItem	512
25.4.8	TNonFreePooledMemManager.EnumerateItems	512
25.4.9	TNonFreePooledMemManager.ItemSize	512
25.5	TPooledMemManager	513
25.5.1	Description	513
25.5.2	Method overview	513
25.5.3	Property overview	513
25.5.4	TPooledMemManager.Clear	513
25.5.5	TPooledMemManager.Create	513
25.5.6	TPooledMemManager.Destroy	514
25.5.7	TPooledMemManager.MinimumFreeCount	514
25.5.8	TPooledMemManager.MaximumFreeCountRatio	514
25.5.9	TPooledMemManager.Count	514
25.5.10	TPooledMemManager.FreeCount	515
25.5.11	TPooledMemManager.AllocatedCount	515
25.5.12	TPooledMemManager.FreedCount	515
26	Reference for unit 'process'	516
26.1	Used units	516
26.2	Overview	516
26.3	Constants, types and variables	516
26.3.1	Types	516
26.3.2	Variables	518
26.4	Procedures and functions	519
26.4.1	CommandToList	519

26.4.2	DetectXTerm	519
26.4.3	RunCommand	519
26.4.4	RunCommandIndir	519
26.5	EProcess	520
26.5.1	Description	520
26.6	TProcess	520
26.6.1	Description	520
26.6.2	Method overview	520
26.6.3	Property overview	521
26.6.4	TProcess.Create	521
26.6.5	TProcess.Destroy	522
26.6.6	TProcess.Execute	522
26.6.7	TProcess.CloseInput	522
26.6.8	TProcess.CloseOutput	523
26.6.9	TProcess.CloseStderr	523
26.6.10	TProcess.Resume	523
26.6.11	TProcess.Suspend	523
26.6.12	TProcess.Terminate	524
26.6.13	TProcess.WaitOnExit	524
26.6.14	TProcess.WindowRect	524
26.6.15	TProcess.Handle	524
26.6.16	TProcess.ProcessHandle	525
26.6.17	TProcess.ThreadHandle	525
26.6.18	TProcess.ProcessID	525
26.6.19	TProcess.ThreadID	526
26.6.20	TProcess.Input	526
26.6.21	TProcess.Output	526
26.6.22	TProcess.Stderr	527
26.6.23	TProcess.ExitStatus	527
26.6.24	TProcess.InheritHandles	527
26.6.25	TProcess.OnForkEvent	528
26.6.26	TProcess.PipeBufferSize	528
26.6.27	TProcess.Active	528
26.6.28	TProcess.ApplicationName	528
26.6.29	TProcess.CommandLine	529
26.6.30	TProcess.Executable	529
26.6.31	TProcess.Parameters	529
26.6.32	TProcess.ConsoleTitle	530
26.6.33	TProcess.CurrentDirectory	530
26.6.34	TProcess.Desktop	531

26.6.35 TProcess.Environment	531
26.6.36 TProcess.Options	531
26.6.37 TProcess.Priority	532
26.6.38 TProcess.StartupOptions	533
26.6.39 TProcess.Running	533
26.6.40 TProcess.ShowWindow	533
26.6.41 TProcess.WindowColumns	534
26.6.42 TProcess.WindowHeight	534
26.6.43 TProcess.WindowLeft	535
26.6.44 TProcess.WindowRows	535
26.6.45 TProcess.WindowTop	535
26.6.46 TProcess.WindowWidth	535
26.6.47 TProcess.FillAttribute	536
26.6.48 TProcess.XTermProgram	536
27 Reference for unit 'rttiutils'	537
27.1 Used units	537
27.2 Overview	537
27.3 Constants, types and variables	537
27.3.1 Constants	537
27.3.2 Types	537
27.3.3 Variables	538
27.4 Procedures and functions	538
27.4.1 CreateStoredItem	538
27.4.2 ParseStoredItem	539
27.4.3 UpdateStoredList	539
27.5 TPropInfoList	539
27.5.1 Description	539
27.5.2 Method overview	540
27.5.3 Property overview	540
27.5.4 TPropInfoList.Create	540
27.5.5 TPropInfoList.Destroy	540
27.5.6 TPropInfoList.Contains	540
27.5.7 TPropInfoList.Find	541
27.5.8 TPropInfoList.Delete	541
27.5.9 TPropInfoList.Intersect	541
27.5.10 TPropInfoList.Count	541
27.5.11 TPropInfoList.Items	542
27.6 TPropsStorage	542
27.6.1 Description	542

27.6.2	Method overview	542
27.6.3	Property overview	542
27.6.4	TPropsStorage.StoreAnyProperty	542
27.6.5	TPropsStorage.LoadAnyProperty	543
27.6.6	TPropsStorage.StoreProperties	543
27.6.7	TPropsStorage.LoadProperties	543
27.6.8	TPropsStorage.LoadObjectsProps	544
27.6.9	TPropsStorage.StoreObjectsProps	544
27.6.10	TPropsStorage.AObject	545
27.6.11	TPropsStorage.Prefix	545
27.6.12	TPropsStorage.Section	545
27.6.13	TPropsStorage.OnReadString	546
27.6.14	TPropsStorage.OnWriteString	546
27.6.15	TPropsStorage.OnEraseSection	546
28	Reference for unit 'simpleipc'	547
28.1	Used units	547
28.2	Overview	547
28.3	Constants, types and variables	547
28.3.1	Resource strings	547
28.3.2	Constants	548
28.3.3	Types	548
28.3.4	Variables	549
28.4	EIPCErrors	549
28.4.1	Description	549
28.5	TIPCCClientComm	549
28.5.1	Description	549
28.5.2	Method overview	549
28.5.3	Property overview	549
28.5.4	TIPCCClientComm.Create	550
28.5.5	TIPCCClientComm.Connect	550
28.5.6	TIPCCClientComm.Disconnect	550
28.5.7	TIPCCClientComm.ServerRunning	551
28.5.8	TIPCCClientComm.SendMessage	551
28.5.9	TIPCCClientComm.Owner	551
28.6	TIPCServerComm	551
28.6.1	Description	551
28.6.2	Method overview	552
28.6.3	Property overview	552
28.6.4	TIPCServerComm.Create	552

28.6.5	TIPCServerComm.StartServer	552
28.6.6	TIPCServerComm.StopServer	552
28.6.7	TIPCServerComm.PeekMessage	553
28.6.8	TIPCServerComm.ReadMessage	553
28.6.9	TIPCServerComm.Owner	553
28.6.10	TIPCServerComm.InstanceID	554
28.7	TSimpleIPC	554
28.7.1	Description	554
28.7.2	Property overview	554
28.7.3	TSimpleIPC.Active	554
28.7.4	TSimpleIPC.ServerID	554
28.8	TSimpleIPCClient	555
28.8.1	Description	555
28.8.2	Method overview	555
28.8.3	Property overview	555
28.8.4	TSimpleIPCClient.Create	555
28.8.5	TSimpleIPCClient.Destroy	555
28.8.6	TSimpleIPCClient.Connect	556
28.8.7	TSimpleIPCClient.Disconnect	556
28.8.8	TSimpleIPCClient.ServerRunning	556
28.8.9	TSimpleIPCClient.SendMessage	557
28.8.10	TSimpleIPCClient.SendStringMessage	557
28.8.11	TSimpleIPCClient.SendStringMessageFmt	557
28.8.12	TSimpleIPCClient.ServerInstance	557
28.9	TSimpleIPCServer	558
28.9.1	Description	558
28.9.2	Method overview	558
28.9.3	Property overview	558
28.9.4	TSimpleIPCServer.Create	558
28.9.5	TSimpleIPCServer.Destroy	559
28.9.6	TSimpleIPCServer.StartServer	559
28.9.7	TSimpleIPCServer.StopServer	559
28.9.8	TSimpleIPCServer.PeekMessage	559
28.9.9	TSimpleIPCServer.GetMessageData	560
28.9.10	TSimpleIPCServer.StringMessage	560
28.9.11	TSimpleIPCServer.MsgType	560
28.9.12	TSimpleIPCServer.MsgData	561
28.9.13	TSimpleIPCServer.InstanceID	561
28.9.14	TSimpleIPCServer.Global	561
28.9.15	TSimpleIPCServer.OnMessage	561

29 Reference for unit 'streamcoll'	562
29.1 Used units	562
29.2 Overview	562
29.3 Procedures and functions	562
29.3.1 ColReadBoolean	562
29.3.2 ColReadCurrency	563
29.3.3 ColReadDateTime	563
29.3.4 ColReadFloat	563
29.3.5 ColReadInteger	563
29.3.6 ColReadString	564
29.3.7 ColWriteBoolean	564
29.3.8 ColWriteCurrency	564
29.3.9 ColWriteDateTime	564
29.3.10 ColWriteFloat	565
29.3.11 ColWriteInteger	565
29.3.12 ColWriteString	565
29.4 EStreamColl	565
29.4.1 Description	565
29.5 TStreamCollection	565
29.5.1 Description	565
29.5.2 Method overview	566
29.5.3 Property overview	566
29.5.4 TStreamCollection.LoadFromStream	566
29.5.5 TStreamCollection.SaveToStream	566
29.5.6 TStreamCollection.Streaming	566
29.6 TStreamCollectionItem	567
29.6.1 Description	567
30 Reference for unit 'streamex'	568
30.1 Used units	568
30.2 Overview	568
30.3 TBidirBinaryObjectReader	568
30.3.1 Description	568
30.3.2 Property overview	568
30.3.3 TBidirBinaryObjectReader.Position	568
30.4 TBidirBinaryObjectWriter	569
30.4.1 Description	569
30.4.2 Property overview	569
30.4.3 TBidirBinaryObjectWriter.Position	569
30.5 TDelphiReader	569

30.5.1	Description	569
30.5.2	Method overview	569
30.5.3	Property overview	570
30.5.4	TDelphiReader.GetDriver	570
30.5.5	TDelphiReader.ReadStr	570
30.5.6	TDelphiReader.Read	570
30.5.7	TDelphiReader.Position	570
30.6	TDelphiWriter	571
30.6.1	Description	571
30.6.2	Method overview	571
30.6.3	Property overview	571
30.6.4	TDelphiWriter.GetDriver	571
30.6.5	TDelphiWriter.FlushBuffer	571
30.6.6	TDelphiWriter.Write	571
30.6.7	TDelphiWriter.WriteStr	572
30.6.8	TDelphiWriter.WriteValue	572
30.6.9	TDelphiWriter.Position	572
31	Reference for unit 'StreamIO'	573
31.1	Used units	573
31.2	Overview	573
31.3	Procedures and functions	573
31.3.1	AssignStream	573
31.3.2	GetStream	574
32	Reference for unit 'syncobjs'	575
32.1	Used units	575
32.2	Overview	575
32.3	Constants, types and variables	575
32.3.1	Constants	575
32.3.2	Types	575
32.4	TCriticalSection	576
32.4.1	Description	576
32.4.2	Method overview	576
32.4.3	TCriticalSection.Acquire	577
32.4.4	TCriticalSection.Release	577
32.4.5	TCriticalSection.Enter	577
32.4.6	TCriticalSection.TryEnter	577
32.4.7	TCriticalSection.Leave	578
32.4.8	TCriticalSection.Create	578
32.4.9	TCriticalSection.Destroy	578

32.5	TEventObject	578
32.5.1	Description	578
32.5.2	Method overview	579
32.5.3	Property overview	579
32.5.4	TEventObject.Create	579
32.5.5	TEventObject.destroy	579
32.5.6	TEventObject.ResetEvent	579
32.5.7	TEventObject.SetEvent	580
32.5.8	TEventObject.WaitFor	580
32.5.9	TEventObject.ManualReset	580
32.6	THandleObject	580
32.6.1	Description	580
32.6.2	Method overview	581
32.6.3	Property overview	581
32.6.4	THandleObject.destroy	581
32.6.5	THandleObject.Handle	581
32.6.6	THandleObject.LastError	581
32.7	TSimpleEvent	581
32.7.1	Description	581
32.7.2	Method overview	582
32.7.3	TSimpleEvent.Create	582
32.8	TSynchroObject	582
32.8.1	Description	582
32.8.2	Method overview	582
32.8.3	TSynchroObject.Acquire	582
32.8.4	TSynchroObject.Release	582
33	Reference for unit 'URIParser'	584
33.1	Overview	584
33.2	Constants, types and variables	584
33.2.1	Types	584
33.3	Procedures and functions	584
33.3.1	EncodeURI	584
33.3.2	FilenameToURI	585
33.3.3	IsAbsoluteURI	585
33.3.4	ParseURI	585
33.3.5	ResolveRelativeURI	586
33.3.6	URIToFilename	586
34	Reference for unit 'zstream'	587
34.1	Used units	587

34.2	Overview	587
34.3	Constants, types and variables	587
34.3.1	Types	587
34.4	Ecompressionerror	588
34.4.1	Description	588
34.5	Edecompressionerror	588
34.5.1	Description	588
34.6	Egzfileerror	588
34.6.1	Description	588
34.7	Ezliberror	588
34.7.1	Description	588
34.8	Tcompressionstream	588
34.8.1	Description	588
34.8.2	Method overview	589
34.8.3	Tcompressionstream.create	589
34.8.4	Tcompressionstream.destroy	589
34.8.5	Tcompressionstream.write	589
34.8.6	Tcompressionstream.flush	590
34.8.7	Tcompressionstream.get_compressionrate	590
34.9	Tcustomzlibstream	590
34.9.1	Description	590
34.9.2	Method overview	590
34.9.3	Tcustomzlibstream.create	590
34.9.4	Tcustomzlibstream.destroy	591
34.10	Tdecompressionstream	591
34.10.1	Description	591
34.10.2	Method overview	591
34.10.3	Tdecompressionstream.create	591
34.10.4	Tdecompressionstream.destroy	592
34.10.5	Tdecompressionstream.read	592
34.10.6	Tdecompressionstream.seek	592
34.10.7	Tdecompressionstream.get_compressionrate	593
34.11	TGZFileStream	593
34.11.1	Description	593
34.11.2	Method overview	593
34.11.3	TGZFileStream.create	593
34.11.4	TGZFileStream.read	594
34.11.5	TGZFileStream.write	594
34.11.6	TGZFileStream.seek	594
34.11.7	TGZFileStream.destroy	595

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

Chapter 1

Reference for unit 'ascii85'

1.1 Used units

Table 1.1: Used units by unit 'ascii85'

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` ([59](#)) 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 ([444](#)) or base64 ([80](#)) units.

1.3 Constants, types and variables

1.3.1 Types

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

Table 1.2: Enumeration values for type TASCII85State

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

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

1.4 TASCII85DecoderStream

1.4.1 Description

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

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

1.4.2 Method overview

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

1.4.3 Property overview

Page	Property	Access	Description
61	BExpectBoundary	rw	Expect character

1.4.4 TASCII85DecoderStream.Create

Synopsis: Create new ASCII 85 decoder stream

Declaration: constructor Create(aStream: TStream)

Visibility: published

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

See also: TASCII85DecoderStream.Destroy (??)

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` (??)

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` (??), `TASCII85DecoderStream.Read` (??), `TASCII85DecoderStream.Destroy` (??)

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` (??), `TASCII85DecoderStream.BExpectBoundary` (??)

1.4.8 TASCII85DecoderStream.Destroy

Synopsis: Clean up instance

Declaration: `destructor Destroy; Override`

Visibility: public

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

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

See also: `TASCII85DecoderStream.Close` (??)

1.4.9 TASCII85DecoderStream.Read

Synopsis: Read data from stream

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

Visibility: public

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

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

1.4.10 TASCII85DecoderStream.Seek

Synopsis: Set stream position

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

Visibility: public

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

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

See also: `TASCII85DecoderStream.Read` (??)

1.4.11 TASCII85DecoderStream.BExpectBoundary

Synopsis: Expect character

Declaration: `Property BExpectBoundary : Boolean`

Visibility: published

Access: Read,Write

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

See also: `ClosedP` (??)

1.5 TASCII85EncoderStream

1.5.1 Description

`TASCII85EncoderStream` is the counterpart to the `TASCII85DecoderStream` (59) decoder stream: what `TASCII85EncoderStream` encodes, can be decoded by `TASCII85DecoderStream` (59).

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

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

See also: `TASCII85EncoderStream.create` (??), `TASCII85DecoderStream` (59)

1.5.2 Method overview

Page	Property	Description
62	Create	Create a new instance of <code>TASCII85EncoderStream</code>
62	Destroy	Flushed the data to the output stream and cleans up the encoder instance.
62	Write	Write data encoded to the destination stream

1.5.3 Property overview

Page	Property	Access	Description
63	Boundary	r	Is a boundary delineator written before and after the data
63	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` ([61](#)), `Width` ([??](#)), `Boundary` ([??](#))

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` ([??](#)), `TASCII85EncoderStream.Boundary` ([??](#))

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 immediatly to the destination stream. Only after the encoding stream is destroyed will the destination stream contain the full data.

See also: `TASCII85EncoderStream.Destroy` ([??](#))

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 (??)`, `Create (??)`

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 (??)`, `Create (??)`

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

See also: `TASCII85DecoderStream` ([59](#))

1.6.2 Method overview

Page	Property	Description
64	<code>Read</code>	Read data from the internal buffer
64	<code>Write</code>	Write data to the internal buffer

1.6.3 Property overview

Page	Property	Access	Description
64	<code>FillCount</code>	<code>r</code>	Number of bytes in buffer
64	<code>Size</code>	<code>r</code>	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` (??), `TASCII85RingBuffer.Read` (??), `TASCII85RingBuffer.Size` (??)

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` (??), `TASCII85RingBuffer.Write` (??), `TASCII85RingBuffer.Size` (??)

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` (??), `TASCII85RingBuffer.Read` (??), `TASCII85RingBuffer.Size` (??)

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` (??)

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` (65) class and it's associated data node class `TAVLTreeNode` (74).

2.3 TAVLTree

2.3.1 Description

`TAVLTree` maintains a balanced AVL tree. The tree consists of `TAVLTreeNode` (74) 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` (??) property.

See also: `TAVLTreeNode` (74)

2.3.2 Method overview

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

2.3.3 Property overview

Page	Property	Access	Description
74	Count	r	Number of nodes in the tree.
74	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` (??) 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` (??).

See also: OnCompare (??), FindKey (??)

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 (??), Find (??)

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 (??)`, `TAVLTree.MoveDataRightMost (??)`

2.3.7 TAVLTree.FindPrecessor

Synopsis:

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

Visibility: public

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

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

See also: `TAVLTree.FindSuccessor (??)`, `TAVLTree.MoveDataLeftMost (??)`

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 (??)`

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 (??)`

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 (??)`, `FindLowest (??)`, `Find (??)`, `FindKey (??)`

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 (??)`, where a custom compare function can be used.

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

See also: `TAVLTree.Find (??)`, `TAVLTree.FindKey (??)`

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 (??)`, `TAVLTree.FindLeftMostKey (??)`, `TAVLTree.FindRightMostKey (??)`

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 (??)`, `TAVLTree.FindLeftMostKey (??)`, `TAVLTree.FindRightMostKey (??)`

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 (??)`, `TAVLTree.FindRightMost (??)`, `TAVLTree.FindRightMostKey (??)`

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 (??)`, `TAVLTree.FindRightMost (??)`, `TAVLTree.FindLeftMostKey (??)`

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 (??)`, `TAVLTree.FindLeftMostKey (??)`, `TAVLTree.FindRightMostSameKey (??)`

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 (??)`, `TAVLTree.FindRightMostKey (??)`, `TAVLTree.FindLeftMostSameKey (??)`

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 (??)`, `TAVLTree.Remove (??)`

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 (76)` 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 (??)`, `TAVLTree.RemovePointer (??)`, `TAVLTree.Clear (??)`

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

See also: `TAVLTree.Delete (??)`, `TAVLTree.RemovePointer (??)`, `TAVLTree.Clear (??)`, `TAVLTree.Find (??)`

2.3.21 TAVLTree.RemovePointer

Synopsis: Remove a pointer item from the list.

Declaration: `procedure RemovePointer(Data: Pointer)`

Visibility: `public`

Description: `Remove` uses `FindPointer` (??) 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` (??), `TAVLTree.Delete` (??), `TAVLTree.Clear` (??)

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` (??)

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` (??)

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` (??) instead.

See also: `TAVLTree.FreeAndClear` (??), `TAVLTree.Delete` (??)

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` (??), `TAVLTree.Delete` (??), `TAVLTree.FreeAndDelete` (??)

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` (??), `TAVLTree.Delete` (??), `TAVLTree.FreeAndClear` (??)

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` (??)

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` (??)

2.3.29 TAVLTree.ReportAsString

Synopsis: Return the tree report as a string

Declaration: `function ReportAsString : string`

Visibility: `public`

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

See also: `TAVLTree.WriteReportToStream` (??)

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

See also: `TBaseAVLTreeNodeManager` (78), `TAVLTreeNodeMemManager` (76)

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

See also: `OnCompare` (??)

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` (??), `TAVLTree.Clean` (??)

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` (75).

See also: `TAVLTreeNodeEnumerator` (75)

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` (??)

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` (65), `TAVLTreeNode.Data` (??)

2.4.2 Method overview

Page	Property	Description
75	<code>Clear</code>	Clears the node's data
75	<code>TreeDepth</code>	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 `TAVLTree` (65). It enumerates all the nodes in the tree.

See also: `TAVLTree` (65)

2.5.2 Method overview

Page	Property	Description
75	<code>Create</code>	Create a new instance of <code>TAVLTreeNodeEnumerator</code>
76	<code>MoveNext</code>	Move to next node in the tree.

2.5.3 Property overview

Page	Property	Access	Description
76	<code>Current</code>	<code>r</code>	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` (??).

See also: `TAVLTree.FindSuccessor` (??)

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` (??)

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` (74), `TAVLTree` (65)

2.6.2 Method overview

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

2.6.3 Property overview

Page	Property	Access	Description
78	<code>Count</code>	r	Number of nodes in the list.
78	<code>MaximumFreeNodeRatio</code>	rw	Maximum amount of free nodes in the list
78	<code>MinimumFreeNode</code>	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` (??)

2.6.5 TAVLTreeNodeMemManager.NewNode

Synopsis: Create a new `TAVLTreeNode` instance

Declaration: `function NewNode : TAVLTreeNode; Override`

Visibility: public

Description: `NewNode` returns a new `TAVLTreeNode` (74) 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` (??)

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` (??), `TAVLTreeNodeMemManager.MaximumFreeNodeRatio` (??)

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` (??)

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` (??)

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` (??)

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` (??), `TAVLTreeNodeMemManager.MinimumFreeNode` (??)

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` (??), `TAVLTreeNodeMemManager.MaximumFreeNodeRatio` (??)

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` (74)) for a `TAVLTree` (65) 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` (76).

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

See also: `TAVLTreeNodeMemManager` (76), `TAVLTree.SetNodeManager` (??), `TAVLTreeNode` (74)

2.7.2 Method overview

Page	Property	Description
79	DisposeNode	Called when the AVL tree no longer needs node
79	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` ([65](#)) when it no longer needs a `TAVLTreeNode` ([74](#)) instance. The manager may decide to re-use the instance for later use instead of destroying it.

See also: `TBaseAVLTreeNodeManager.NewNode` ([??](#)), `TAVLTree.Delete` ([??](#)), `TAVLTreeNode` ([74](#))

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` ([65](#)) when it needs a new node in `TAVLTree.Add` ([??](#)). It must be implemented by descendants to return a new `TAVLTreeNode` ([74](#)) instance.

See also: `TBaseAVLTreeNodeManager.DisposeNode` ([??](#)), `TAVLTree.Add` ([??](#)), `TAVLTreeNode` ([74](#))

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` (84) for encoding, and `TBase64DecodingStream` (81) 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, bdMIME)
```

Table 3.2: Enumeration values for type `TBase64DecodingMode`

Value	Explanation
<code>bdmMIME</code>	MIME encoding
<code>bdmStrict</code>	Strict encoding

`TBase64DecodingMode` determines the decoding algorithm used by `TBase64DecodingStream` (81). 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` (81) 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` (81) to do this.

See also: `DecodeStringBase64` (81), `TBase64DecodingStream` (81)

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` (84) to do this.

See also: `DecodeStringBase64` (81), `TBase64EncodingStream` (84)

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-seeking capability.

See also: [TBase64EncodingStream \(84\)](#)

3.6.2 Method overview

Page	Property	Description
82	Create	Create a new instance of the <code>TBase64DecodingStream</code> class
82	Read	Read and decrypt data from the source stream
82	Reset	Reset the stream
83	Seek	Set stream position.

3.6.3 Property overview

Page	Property	Access	Description
83	EOF	r	
83	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 \(??\)](#), [TBase64DecodingMode \(80\)](#)

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 \(??\)](#), [TBase64DecodingStream.Read \(??\)](#)

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 (??)`, `TBase64DecodingStream.Seek (??)`, `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 (??)`, `TBase64DecodingStream.Write (??)`, `TBase64EncodingStream.Seek (??)`, `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 (81)`

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 (??) 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 (81)

3.7.2 Method overview

Page	Property	Description
84	Destroy	Remove a TBase64EncodingStream instance from memory
84	Flush	Flush the remaining bytes to the output stream.
85	Seek	Position the stream
85	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 (??)

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 (??)

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` (??), `TBase64EncodingStream.Read` (??), `TBase64DecodingStream.Write` (??), `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` (??), `TBase64EncodingStream.Write` (??), `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 (87) to handle blowfish encryption/decryption of memory buffers, and 2 TStream (??) descendents TBlowFishDeCryptStream (88) which decrypts any data that is read from it on the fly, as well as TBlowFishEnCryptStream (89) 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 (87) array.

TBFBlock = Array[0..1] of LongInt

TBFBlock is the basic data structure used by the encrypting/decrypting routines in TBlowFish (87), TBlowFishDeCryptStream (88) and TBlowFishEnCryptStream (89). 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 (87), TBlowFishDeCryptStream (88) and TBlowFishEnCryptStream (89) 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 (91), TBlowFishEncryptStream (89) and TBlowFishDecryptStream (88) classes to report errors.

See also: TBlowFishStream (91), TBlowFishEncryptStream (89), TBlowFishDecryptStream (88)

4.5 TBlowFish

4.5.1 Description

TBlowFish is a simple class that can be used to encrypt/decrypt a single TBFBlock (87) data block with the Encrypt (??) and Decrypt (??) calls. It is used internally by the TBlowFishEnCryptStream (89) and TBlowFishDeCryptStream (88) classes to encrypt or decrypt the actual data.

See also: TBlowFishEnCryptStream (89), TBlowFishDeCryptStream (88)

4.5.2 Method overview

Page	Property	Description
87	Create	Create a new instance of the TBlowFish class
88	Decrypt	Decrypt a block
88	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 (??) and Decrypt (??).

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

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 (87) specified when the `TBlowFish` instance was created.

See also: `TBlowFishKey` (87), `Decrypt` (??), `Create` (??)

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

See also: `TBlowFishKey` (87), `Encrypt` (??), `Create` (??)

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` (89) stream must be used.

See also: `Create` (??), `TBlowFishEncryptStream` (89)

4.6.2 Method overview

Page	Property	Description
88	Read	Read data from the stream
89	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` (??), `TBlowFishEncryptStream` (89)

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` (87) exception will be raised.

See also: `Read` (??), `EBlowFishError` (87)

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` (88) stream must be used.

See also: `Create` (??), `TBlowFishDeCryptStream` (88)

4.7.2 Method overview

Page	Property	Description
89	<code>Destroy</code>	Free the <code>TBlowFishEncryptStream</code>
90	<code>Flush</code>	Flush the encryption buffer
90	<code>Seek</code>	Set the position in the stream
90	<code>Write</code>	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 (??)`, `Create (??)`

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 (??)` mechanism can be used to write the remaining bytes.

See also: `TBlowFishEncryptStream.Read (??)`

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 (87)` exception.

See also: `TBlowFishEncryptStream.Write (??)`

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 (??)`, `TBFBlock (87)`

4.8 TBlowFishStream

4.8.1 Description

TBlowFishStream is an abstract class which is used as a parent class for TBlowFishEncryptStream (89) and TBlowFishDecryptStream (88). It simply provides a constructor and storage for a TBlowFish (87) 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 (89), TBlowFishDecryptStream (88), TBlowFish (87)

4.8.2 Method overview

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

4.8.3 Property overview

Page	Property	Access	Description
92	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 (87) 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 (89), TBlowFishDecryptStream (88), TBlowFish (87)

4.8.5 TBlowFishStream.Destroy

Synopsis: Destroy the TBlowFishStream instance.

Declaration: destructor Destroy; Override

Visibility: public

Description: Destroy cleans up the internal TBlowFish (87) instance.

See also: Create (??), TBlowFish (87)

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` (87) 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` (89) and `TBlowFishDecryptStream` (88).

See also: `TBlowFishEncryptStream` (89), `TBlowFishDecryptStream` (88), `TBlowFish` (87)

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 (96) which is for reading only, and TWriteBufStream (97) 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 (96) and TWriteBufStream (97) 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: [TReadStream \(96\)](#), [TWriteBufStream \(97\)](#)

5.4.2 Method overview

Page	Property	Description
94	Create	Create a new <code>TBufStream</code> instance.
94	Destroy	Destroys the <code>TBufStream</code> instance

5.4.3 Property overview

Page	Property	Access	Description
95	Buffer	r	The current buffer
95	BufferPos	r	Current buffer position.
95	BufferSize	r	Amount of data in the buffer
95	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` ([93](#)) is used as the capacity.

An instance of `TBufStream` should never be instantiated directly. Instead, an instance of `TReadStream` ([96](#)) or `TWriteBufStream` ([97](#)) should be created.

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

See also: `TBufStream.Destroy` ([??](#)), [TReadStream \(96\)](#), [TWriteBufStream \(97\)](#)

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` ([??](#)), [TReadStream \(96\)](#), [TWriteBufStream \(97\)](#)

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 (??)`, `TBufStream.Capacity (??)`, `TBufStream.BufferSize (??)`

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 (??)`, `TBufStream.Buffer (??)`, `TBufStream.BufferSize (??)`, `TBufStream.BufferPos (??)`

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 (??)`, `TBufStream.Buffer (??)`, `TBufStream.BufferSize (??)`, `TBufStream.Capacity (??)`

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 (??)`.

See also: `TBufStream.Create (??)`, `TBufStream.Buffer (??)`, `TBufStream.BufferPos (??)`, `TBufStream.Capacity (??)`

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` (93), `TWriteBufStream` (97)

5.5.2 Method overview

Page	Property	Description
96	Read	Reads data from the stream
96	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` (??), `TReadBufStream.Read` (??), `TReadBufStream.Write` (??)

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` (??), `TReadBufStream.Read` (??)

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` (93), `TReadBufStream` (96)

5.6.2 Method overview

Page	Property	Description
97	<code>Destroy</code>	Remove the <code>TWriteBufStream</code> instance from memory
97	<code>Seek</code>	Set stream position.
97	<code>Write</code>	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` (??).

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

See also: `Create` (??), `TBufStream.Destroy` (??)

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` (??), `TWriteBufStream.Read` (??), `TReadBufStream.Seek` (??)

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` (??), `TWriteBufStream.Read` (??), `TReadBufStream.Write` (??)

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` (100) record.

```
PCacheSlotArray = ^TCacheSlotArray
```

Pointer to `TCacheSlotArray` (100) array

```
TCacheSlot = record
```

```

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

```

TCacheSlot is internally used by the TCache (100) 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 (99) 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 (??) 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 (??) property offers indexed access to the array of items. When accessing the array through this property, the MRUSlot (??) property is updated.

6.5.2 Method overview

Page	Property	Description
101	Add	Add a data element to the list.
102	AddNew	Add a new item to the list.
101	Create	Create a new cache class.
101	Destroy	Free the TCache class from memory
102	FindSlot	Find data pointer in the list
102	IndexOf	Return index of a data pointer in the list.
103	Remove	Remove a data item from the list.

6.5.3 Property overview

Page	Property	Access	Description
103	Data	rw	Indexed access to data items
104	LRUSlot	r	Last used item
103	MRUSlot	rw	Most recent item slot.
105	OnFreeSlot	rw	Event called when a slot is freed
104	OnIsDataEqual	rw	Event to compare 2 items.
104	SlotCount	rw	Number of slots in the list
104	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` (??)

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` (??)

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 (??)` 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 (??)`, `TCache.FindSlot (??)`, `TCache.IndexOf (??)`, `TCache.Data (??)`, `TCache.MRUSlot (??)`

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 (??)` 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 (??)`, `TCache.FindSlot (??)`, `TCache.IndexOf (??)`, `TCache.Data (??)`, `TCache.MRUSlot (??)`

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 (??)` event must be set.

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

See also: `TCache.IndexOf (??)`, `TCache.Add (??)`, `TCache.OnIsDataEqual (??)`

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 (??)` event must be set.

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

See also: `TCache.FindSlot (??)`, `TCache.Add (??)`, `TCache.OnIsDataEqual (??)`

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 (??)`

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

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

See also: `TCache.MRUSlot (??)`

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 (??)` property, or when an item is added to the list with `Add (??)` or `AddNew (??)`

See also: `TCache.Add (??)`, `TCache.AddNew (??)`, `TCache.Data (??)`, `TCache.LRUSlot (??)`

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 (??), TCache.AddNew (??), TCache.Data (??), TCache.MRUSlot (??)

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 (??) is called.

See also: TCache.Create (??), TCache.Data (??), TCache.Slots (??)

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 TCacheSlot records in the list. Accessing the records directly does not change their position in the MRU list.

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

See also: TCache.Data (??), TCache.SlotCount (??)

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 (??)` and `IndexOf (??)` to compare items when looking for a particular item. These functions are called by the `Add (??)` 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 (??)`, `TCache.IndexOf (??)`, `TCache.Add (??)`

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 (??)`, `TCache.AddNew (??)`, `TCache.SlotCount (??)`

Chapter 7

Reference for unit 'contrns'

7.1 Used units

Table 7.1: Used units by unit 'contrns'

Name	Page
Classes	??
System	??
sysutils	??

7.2 Overview

The `contrns` unit implements various general-purpose classes:

Object lists lists that manage objects instead of pointers, and which automatically dispose of the objects.

Component lists lists that manage components instead of pointers, and which automatically dispose the components.

Class lists lists that manage class pointers instead of pointers.

Stacks Stack classes to push/pop pointers or objects

Queues Classes to manage a FIFO list of pointers or objects

Hash lists General-purpose Hash lists.

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 (107)" type.

```
PHashItem = ^THashItem
```

PHashItem is a pointer type, pointing to the THashItem (109) record.

```
PHashItemList = ^THashItemList
```

PHashItemList is a pointer to the THashItemList (109). It's used in the TFPHashList (126) as a pointer to the memory area containing the hash item records.

```
PHashTable = ^THashTable
```

PHashTable is a pointer to the THashTable (109). It's used in the TFPHashList (126) 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 (117) class. It is a container for TBucketItem (108) records. It should never be used directly.

```
TBucketArray = Array of TBucket
```

Array of TBucket (107) records.

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

TBucketItem is a record used for internal use in TCustomBucketList (117). 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 (110).

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

TBucketProc is the prototype for the TCustomBucketList.Foreach (??) 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 (108) prototype.

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

TBucketProcObject is the prototype for the TCustomBucketList.Foreach (??) 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 (108) prototype.

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

TDataIteratorMethod is a callback prototype for the TFPDataHashTable.Iterate (??) 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 (107) type. It's used in the TFPHashList (126) 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 (107) class.

```
THTCustomNodeClass = Class of THTCustomNode
```

THTCustomNodeClass was used by TFPCustomHashTable (119) 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 (125) 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 TFPObjectList.ForEachCall (??) 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 (??)` 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 (??)` 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 (119)` hash class. It's Robert Sedgwick's "Algorithms in C" hash function.

Errors: None.

See also: `TFPCustomHashTable (119)`

7.5 EDuplicate

7.5.1 Description

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

See also: `TFPCustomHashTable.Add (??)`

7.6 EKeyNotFound

7.6.1 Description

Exception raised when a key is not found.

See also: `TFPCustomHashTable.Delete (??)`

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 \(117\)](#)

7.7.2 Method overview

Page	Property	Description
111	Create	Create a new <code>TBucketList</code> 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 \(108\)](#)

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 \(114\)](#), [TObjectList \(158\)](#)

7.8.2 Method overview

Page	Property	Description
112	Add	Add a new class pointer to the list.
112	Extract	Extract a class pointer from the list.
113	First	Return first non-nil class pointer
112	IndexOf	Search for a class pointer in the list.
113	Insert	Insert a new class pointer in the list.
113	Last	Return last non- <code>Nil</code> class pointer
112	Remove	Remove a class pointer from the list.

7.8.3 Property overview

Page	Property	Access	Description
113	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 (??), #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 (??), #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 (??), #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 it's position if it was found, or -1 if it was not found in the list.

Errors: None.

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

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` (??), `TClassList.Pack` (??)

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` (??), `TClassList.Pack` (??)

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` (??), `TClassList.Remove` (??)

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` (158) 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` (158), `TComponentList` removes any `TComponent` from the list if the `TComponent` instance was freed externally. It uses the `FreeNotification` mechanism for this.

See also: `#rtl.classes.TList` (??), `TFPObjectList` (145), `TObjectList` (158), `TClassList` (111)

7.9.2 Method overview

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

7.9.3 Property overview

Page	Property	Access	Description
116	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` (158), `#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 it's ancestors, so it only accepts `TComponent` instances. It introduces no new behaviour.

The function returns the index at which the component was added.

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

See also: `TObjectList.Add` (??)

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` (158) so only `TComponent` descendents can be extracted. It introduces no new behaviour.

`Extract` returns the instance that was extracted, or `Nil` if no instance was found.

See also: `TComponentList.Remove` (??), `TObjectList.Extract` (??)

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 it's items, it also destroys it. It returns the index of the item that was removed, or -1 if no item was removed.

`Remove` simply overrides the implementation in `TObjectList` (158) so it only accepts `TComponent` descendents. It introduces no new behaviour.

Errors: None.

See also: `TComponentList.Extract` (??), `TObjectList.Remove` (??)

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 it's position in the list. The position is zero-based. If no instance is found, -1 is returned.

`IndexOf` just overrides the implementation of the parent class so it accepts only `TComponent` instances. It introduces no new behaviour.

Errors: None.

See also: `TObjectList.IndexOf` (??)

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` (??), `TObjectList.First` (??)

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` (??), `TObjectList.Last` (??)

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` (??), `TComponentList.Add` (??), `TComponentList.Remove` (??)

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` (??)

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 (110). 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 (117).

See also: TBucketList (110)

7.10.2 Method overview

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

7.10.3 Property overview

Page	Property	Access	Description
119	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` (??)

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` (??), `TCustomBucketList.Clear` (??)

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` (??), `TCustomBucketList.Clear` (??)

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` (??).

See also: `TCustomBucketList.Find` (??)

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` (??)

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: Foreach loops over all items in the list and calls AProc, passing it in turn each item in the list.

AProc exists in 2 variants: one which is a simple procedure, and one which is a method. In the case of the simple procedure, the AInfo argument is passed as well in each call to AProc.

The loop stops when all items have been processed, or when the AContinue argument of AProc contains False on return.

The result of the function is True if all items were processed, or False if the loop was interrupted with a AContinue return of False.

Errors: None.

See also: TCustomBucketList.Data (??)

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 (??)

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 (??), TCustomBucketList.Exists (??)

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` (110).

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` (153), `TFPObjectList` (145), `RSHash` (110)

7.11.2 Method overview

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

7.11.3 Property overview

Page	Property	Access	Description
124	<code>AVGChainLen</code>	r	Average chain length
123	<code>Count</code>	r	Number of items in the hash table.
125	<code>Density</code>	r	Number of filled slots
122	<code>HashFunction</code>	rw	Hash function currently in use
123	<code>HashTable</code>	r	Hash table instance
123	<code>HashTableSize</code>	rw	Size of the hash table
124	<code>LoadFactor</code>	r	Fraction of count versus size
124	<code>MaxChainLength</code>	r	Maximum chain length
124	<code>NumberOfCollisions</code>	r	Number of extra items
123	<code>VoidSlots</code>	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` (110)

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

See also: `CreateWith` (??)

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` (??)

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` (??), `Create` (??), `Create` (??), `THTCustomNode.Data` (??)

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` (??)

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` (??)

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` (??), `TFPCustomHashTable.Add` (??)

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` (153) 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` (??) method should be used instead.

Errors: None.

See also: `Add` (??), `Delete` (??)

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` (??), `TFPCustomHashTable.HashTableSize` (??), `TFPCustomHashTable.AVGChainLen` (??), `TFPCustomHashTable.MaxChainLength` (??)

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` (??). 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` (??), `HashTableSize` (??)

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 (??)`, `TFPCustomHashTable.HashTableSize (??)`, `TFPCustomHashTable.AVGChainLen (??)`, `TFPCustomHashTable.MaxChainLength (??)`

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 (??)`, `TFPCustomHashTable.Count (??)`, `TFPCustomHashTable.AVGChainLen (??)`, `TFPCustomHashTable.MaxChainLength (??)`, `TFPCustomHashTable.VoidSlots (??)`, `TFPCustomHashTable.Density (??)`

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` (145)) used for the hash table. Each element in this table is again a `TFPObjectList` (145) 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 (??)`, `TFPCustomHashTable.Count (??)`, `TFPCustomHashTable.AVGChainLen (??)`, `TFPCustomHashTable.MaxChainLength (??)`, `TFPCustomHashTable.LoadFactor (??)`, `TFPCustomHashTable.Density (??)`, `TFPCustomHashTable.NumberOfCollisions (??)`

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 (??), TFPCustomHashTable.Count (??), TFPCustomHashTable.AVGChainLen (??), TFPCustomHashTable.MaxChainLength (??), TFPCustomHashTable.VoidSlots (??), TFPCustomHashTable.Density (??), TFPCustomHashTable.NumberOfCollisions (??)

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 (??), TFPCustomHashTable.Count (??), TFPCustomHashTable.LoadFactor (??), TFPCustomHashTable.MaxChainLength (??), TFPCustomHashTable.VoidSlots (??), TFPCustomHashTable.Density (??), TFPCustomHashTable.NumberOfCollisions (??)

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 (??), TFPCustomHashTable.Count (??), TFPCustomHashTable.LoadFactor (??), TFPCustomHashTable.AvgChainLength (??), TFPCustomHashTable.VoidSlots (??), TFPCustomHashTable.Density (??), TFPCustomHashTable.NumberOfCollisions (??)

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` (??), `TFPCustomHashTable.Count` (??), `TFPCustomHashTable.LoadFactor` (??), `TFPCustomHashTable.AvgChainLength` (??), `TFPCustomHashTable.VoidSlots` (??), `TFPCustomHashTable.Density` (??)

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` (??), `TFPCustomHashTable.Count` (??), `TFPCustomHashTable.LoadFactor` (??), `TFPCustomHashTable.AvgChainLength` (??), `TFPCustomHashTable.VoidSlots` (??), `TFPCustomHashTable.Density` (??)

7.12 TFPDataHashTable

7.12.1 Description

`TFPDataHashTable` is a `TFPCustomHashTable` (119) 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` (143), or for string data, `TFPStringHashTable` (152) is more suitable. The data pointers are exposed with their keys through the `Items` (??) property.

See also: `TFPObjectHashTable` (143), `TFPStringHashTable` (152), `Items` (??)

7.12.2 Method overview

Page	Property	Description
125	<code>Add</code>	Add a data pointer to the list.

7.12.3 Property overview

Page	Property	Access	Description
126	<code>Items</code>	rw	Key-based access to the items in the table

7.12.4 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 (??)

7.12.5 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 (??)

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 (119) class can be used. If a hash class for objects is needed instead of pointers, the TFPHashObjectList (136) class can be used.

See also: TFPCustomHashTable (119), TFPHashObjectList (136), TFPDataHashTable (125), TFPStringHashTable (152)

7.13.2 Method overview

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

7.13.3 Property overview

Page	Property	Access	Description
132	Capacity	rw	Capacity of the list.
132	Count	rw	Current number of elements in the list.
133	Items	rw	Indexed array with pointers
133	List	r	Low-level hash list
133	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` (??)

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 (??)`, `TFPHashList.Clear (??)`

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 (??)`, `TFPHashList.Remove (??)`, `TFPHashList.Delete (??)`

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 (??)`, `TFPHashList.Remove (??)`, `TFPHashList.Delete (??)`, `TFPHashList.Add (??)`

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 (??)`, `TFPHashList.Find (??)`, `TFPHashList.FindIndexOf (??)`, `TFPHashList.FindWithHash (??)`

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 (??)`, `TFPHashList.Find (??)`, `TFPHashList.FindIndexOf (??)`, `TFPHashList.FindWithHash (??)`

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 (??)``TFPHashList.Remove (??)``TFPHashList.Add (??)`

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 (??)`

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` (??), `TFPHashList.Remove` (??), `TFPHashList.Clear` (??)

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` (??), `TFPHashList.NameOfIndex` (??), `TFPHashList.Find` (??), `TFPHashList.FindIndexOf` (??), `TFPHashList.FindWithHash` (??)

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` (??), `TFPHashList.NameOfIndex` (??), `TFPHashList.IndexOf` (??), `TFPHashList.FindIndexOf` (??), `TFPHashList.FindWithHash` (??)

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` (??), `TFPHashList.NameOfIndex` (??), `TFPHashList.IndexOf` (??), `TFPHashList.Find` (??), `TFPHashList.FindWithHash` (??)

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 (??)`, `TFPHashList.NameOfIndex (??)`, `TFPHashList.IndexOf (??)`, `TFPHashList.Find (??)`, `TFPHashList.FindIndexOf (??)`

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 (??)`, `TFPHashList.Clear (??)`, `TFPHashList.Extract (??)`

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 (??)`

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:

HashSizeSize of the hash table

HashMeanMean hash value

HashStdDevStandard deviation of hash values

ListSizeSize and capacity of the list

StringSizeSize and capacity of key strings

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 (??)`, `Items (??)`

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 (??)`, `Items (??)`

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 (??).

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

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

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 (109). It should not be used directly.

See also: `Strs` (??), `THashItemList` (109)

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` (??)

7.14 TFPHashObject

7.14.1 Description

`TFPHashObject` is a `TObject` descendent which is aware of the `TFPHashObjectList` (136) 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` (??), `TFPHashObject.ChangeOwner` (??), `TFPHashObject.ChangeOwnerAndName` (??)

7.14.2 Method overview

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

7.14.3 Property overview

Page	Property	Access	Description
135	Hash	r	Hash value
135	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` ([136](#)) hash list. It also has no name when created in this way.

See also: `TFPHashObject.Name` ([??](#)), `TFPHashObject.ChangeOwner` ([??](#)), `TFPHashObject.ChangeOwnerAndName` ([??](#))

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` ([136](#)).

See also: `CreateNotOwned` ([??](#)), `TFPHashObject.ChangeOwner` ([??](#)), `TFPHashObject.Name` ([??](#))

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` ([??](#)), `Name` ([??](#))

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` (??), `Name` (??)

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` (??), `ChangeOwnerAndName` (??), `Name` (??)

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` (??), `ChangeOwnerAndName` (??)

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` (??)

7.15 TFPHashObjectList

7.15.1 Method overview

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

7.15.2 Property overview

Page	Property	Access	Description
141	Capacity	rw	Capacity of the list.
142	Count	rw	Current number of elements in the list.
142	Items	rw	Indexed array with object instances
142	List	r	Low-level hash list
142	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` (??), `TFPHashObjectList.OwnsObjects` (??)

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 (??)`, `TFPHashObjectList.Clear (??)`

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

Errors: None.

See also: `TFPHashObjectList.Extract (??)`, `TFPHashObjectList.Remove (??)`, `TFPHashObjectList.Delete (??)`, `TFPHashObjectList.Add (??)`

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 (??)`, `TFPHashObjectList.Remove (??)`, `TFPHashObjectList.Delete (??)`

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 (??)`, `TFPHashObjectList.Find (??)`, `TFPHashObjectList.FindIndexOf (??)`, `TFPHashObjectList.FindWithHash (??)`

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` (??), `TFPHashObjectList.Find` (??), `TFPHashObjectList.FindIndexOf` (??), `TFPHashObjectList.FindWithHash` (??)

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

See also: `TFPHashObjectList.Extract` (??), `TFPHashObjectList.Remove` (??), `TFPHashObjectList.Add` (??), `OwnsObjects` (??)

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` (??)

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` (??).

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

See also: `TFPHashObjectList.Delete` (??), `TFPHashObjectList.Remove` (??), `TFPHashObjectList.Clear` (??)

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

See also: `TFPHashObjectList.Delete` (??), `TFPHashObjectList.Clear` (??), `TFPHashObjectList.Extract` (??)

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` (??), `TFPHashObjectList.NameOfIndex` (??), `TFPHashObjectList.Find` (??), `TFPHashObjectList.FindIndexOf` (??), `TFPHashObjectList.FindWithHash` (??)

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` (??), `TFPHashObjectList.NameOfIndex` (??), `TFPHashObjectList.IndexOf` (??), `TFPHashObjectList.FindIndexOf` (??), `TFPHashObjectList.FindWithHash` (??)

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 (??)`, `TFPHashObjectList.NameOfIndex (??)`, `TFPHashObjectList.IndexOf (??)`, `TFPHashObjectList.Find (??)`, `TFPHashObjectList.FindWithHash (??)`

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 (??)`, `TFPHashObjectList.NameOfIndex (??)`, `TFPHashObjectList.IndexOf (??)`, `TFPHashObjectList.Find (??)`, `TFPHashObjectList.FindIndexOf (??)`

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` (??)

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:

HashSizeSize of the hash table

HashMeanMean hash value

HashStdDevStandard deviation of hash values

ListSizeSize and capacity of the list

StringSizeSize and capacity of key strings

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` (??), `Items` (??)

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 (??)`, `Items (??)`

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 (??)`

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 (??)`.

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

See also: `Capacity (??)`, `Count (??)`

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 ([126](#)). It should not be used directly.

See also: `TFPHashList (126)`

7.16 TFPObjectHashTable

7.16.1 Description

TFPStringHashTable is a TFPCustomHashTable (119) 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 (152), or for arbitrary pointer data, TFPDataHashTable (125) is more suitable. The objects are exposed with their keys through the Items (??) property.

See also: TFPStringHashTable (152), TFPDataHashTable (125), TFPObjectHashTable.Items (??)

7.16.2 Method overview

Page	Property	Description
144	Add	Add a new object to the hash table
143	Create	Create a new instance of TFPObjectHashTable
143	CreateWith	Create a new hash table with given size and hash function

7.16.3 Property overview

Page	Property	Access	Description
144	Items	rw	Key-based access to the objects
144	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 (??) 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 (??), TFPObjectHashTable.CreateWith (??), TFPObjectHashTable.Items (??)

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 (??) 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: `TFObjectHashTable.OwnsObjects` (??), `TFObjectHashTable.Create` (??), `TFObjectHashTable.Items` (??)

7.16.6 TFObjectHashTable.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: `TFObjectHashTable.Items` (??)

7.16.7 TFObjectHashTable.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: `TFObjectHashTable.Add` (??)

7.16.8 TFObjectHashTable.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: `TFObjectHashTable.Create` (??), `TFObjectHashTable.Items` (??)

7.17 TFObjectList

7.17.1 Description

`TFObjectList` 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` (158), `TFObjectList` 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` (158)

7.17.2 Method overview

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

7.17.3 Property overview

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

7.17.4 TFObjectList.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` (??), `TFPObjectList.OwnsObjects` (??), `TObjectList` ([158](#))

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` (??) is `True`.

See also: `TFPObjectList.OwnsObjects` (??), `TObjectList.Create` (??)

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` (??) is `True`.

See also: `TObjectList.Destroy` (??)

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` (??) 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` (??), `TFPObjectList.Delete` (??)

7.17.8 TFObjectList.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` (??) is `True`, the object is also freed.

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

See also: `TFObjectList.Remove` (??), `TFObjectList.Extract` (??), `TFObjectList.OwnsObjects` (??), `TFObjectList.Add` (??), `TFObjectList.Clear` (??)

7.17.9 TFObjectList.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: `TFObjectList.Add` (??), `TFObjectList.Delete` (??)

7.17.10 TFObjectList.Expand

Synopsis: Expand the capacity of the list.

Declaration: `function Expand : TFObjectList`

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: `TFObjectList.Pack` (??), `TFObjectList.Clear` (??), `#rtl.classes.tfplist.expand` (??)

7.17.11 TFObjectList.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: `TFObjectList.Pack (??)`, `TFObjectList.Clear (??)`, `TFObjectList.Remove (??)`, `TFObjectList.Delete (??)`

7.17.12 TFObjectList.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 (??)` 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: `TFObjectList.Pack (??)`, `TFObjectList.Clear (??)`, `TFObjectList.Delete (??)`, `TFObjectList.Extract (??)`

7.17.13 TFObjectList.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: `TFObjectList.Items (??)`, `TFObjectList.Remove (??)`, `TFObjectList.Extract (??)`

7.17.14 TFObjectList.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: `TFObjectList.IndexOf (??)`

7.17.15 TFObjectList.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: `TFObjectList.Add (??)`, `TFObjectList.Delete (??)`

7.17.16 TFObjectList.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: `TFObjectList.Last (??)`, `TFObjectList.Pack (??)`

7.17.17 TFObjectList.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: `TFObjectList.First (??)`, `TFObjectList.Pack (??)`

7.17.18 TFObjectList.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 (??)`, 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 (??)`, `TFPObjectList.Delete (??)`, `TFPObjectList.Insert (??)`

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 (??)`, `TFPObjectList.Last (??)`

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 TFObjectList.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` (110), `TObjectListCallback` (109)

7.17.23 TFObjectList.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` (??).

See also: `TFObjectList.Count` (??)

7.17.24 TFObjectList.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: `TFObjectList.Capacity` (??)

7.17.25 TFObjectList.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 (??)`, `TFPObjectList.Delete (??)`, `TFPObjectList.Remove (??)`, `TFPObjectList.Clear (??)`

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 (??)`

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 (119)` descendent which stores simple strings together with the keys. In case the data associated with the keys are objects, it's better to use `TFPObjectHashTable (143)`, or for arbitrary pointer data, `TFPDataHashTable (125)` is more suitable. The strings are exposed with their keys through the `Items (??)` property.

See also: `TFPObjectHashTable (143)`, `TFPDataHashTable (125)`, `Items (??)`

7.18.2 Method overview

Page	Property	Description
153	<code>Add</code>	Add a new string to the hash list

7.18.3 Property overview

Page	Property	Access	Description
153	Items	rw	Key based access to the strings in the hash table

7.18.4 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` (??)

7.18.5 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` (??)

7.19 THTCustomNode

7.19.1 Description

`THTCustomNode` is used by the `TFPCustomHashTable` ([119](#)) class to store the keys and associated values.

See also: `TFPCustomHashTable` ([119](#))

7.19.2 Method overview

Page	Property	Description
154	CreateWith	Create a new instance of <code>THTCustomNode</code>
154	HasKey	Check whether this node matches the given key.

7.19.3 Property overview

Page	Property	Access	Description
154	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` (119) class when needed.

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

See also: `TFPCustomHashTable` (119)

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` (??)

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` (??)

7.20 THTDataNode

7.20.1 Description

`THTDataNode` is used by `TFPDataHashTable` (125) 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` (125), `THTObjectNode` (155), `THTStringNode` (156)

7.20.2 Property overview

Page	Property	Access	Description
155	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 ([153](#)) descendent which holds the data in the TFPObjectHashTable ([143](#)) hash table. It exposes a data string.

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

See also: TFPObjectHashTable ([143](#))

7.21.2 Property overview

Page	Property	Access	Description
155	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 (??)

See also: TFPObjectHashTable ([143](#)), TFPObjectHashTable.Items (??), THTOwnedObjectNode ([155](#))

7.22 THTOwnedObjectNode

7.22.1 Description

THTOwnedObjectNode is used instead of THTObjectNode ([155](#)) in case TFPObjectHashTable ([143](#)) owns it's objects. When this object is destroyed, the associated data object is also destroyed.

See also: TFPObjectHashTable ([143](#)), THTObjectNode ([155](#)), TFPObjectHashTable.OwnsObjects (??)

7.22.2 Method overview

Page	Property	Description
156	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` ([155](#)), `TFPObjectHashTable.OwnsObjects` (??)

7.23 THTStringNode

7.23.1 Description

`THTStringNode` is a `THTCustomNode` ([153](#)) descendent which holds the data in the `TFPStringHashTable` ([152](#)) 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` ([152](#))

7.23.2 Property overview

Page	Property	Access	Description
156	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` (??)

See also: `TFPStringHashTable` ([152](#))

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` ([156](#))

7.24.2 Method overview

Page	Property	Description
157	Add	Add an object to the list
157	Remove	Remove an object from the list

7.24.3 Property overview

Page	Property	Access	Description
157	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 (??), TObjectBucketList.Remove (??)

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 (??), TObjectBucketList.Data (??)

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 (??)

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` (145), `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` (145), `TComponentList` (114), `TClassList` (111)

7.25.2 Method overview

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

7.25.3 Property overview

Page	Property	Access	Description
161	Items	rw	Indexed access to the elements of the list.
161	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` (??), `TObjectList.OwnsObjects` (??), `TFPObjectList` (145)

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 (??), #rtl.classes.TList.Delete (??), TObjectList.Extract (??), TObjectList.Remove (??)

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 (??), Extract does not free the extracted element if OwnsObjects (??) 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 (??)

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 (??) 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 (??)

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 (??)

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 (??)`

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 (??)`, `TObjectList.Remove (??)`

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 (??)`, `TObjectList.Pack (??)`

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 (??)`, `TObjectList.Pack (??)`

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 (??)`, `TObjectList.Delete (??)`, `TObjectList.Remove (??)`, `TObjectList.Clear (??)`

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
162	<code>Peek</code>	Look at the first object in the queue.
162	<code>Pop</code>	Pop the first element off the queue
161	<code>Push</code>	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 (??)`, `TObjectQueue.Peek (??)`

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` (??), `TObjectQueue.Peek` (??)

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` (??), `TObjectQueue.Pop` (??)

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` ([163](#)), `TStack` ([166](#)), `TQueue` ([166](#)), `TObjectQueue` ([161](#))

7.27.2 Method overview

Page	Property	Description
163	<code>Peek</code>	Look at the top object in the stack.
163	<code>Pop</code>	Pop the top object of the stack.
162	<code>Push</code>	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` (??), `TObjectStack.Peek` (??)

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` (??), `TObjectStack.Peek` (??)

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` (??), `TObjectStack.Pop` (??)

7.28 `TOrderedList`

7.28.1 Description

`TOrderedList` provides the base class for `TQueue` (166) and `TStack` (166). 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` (166), `TStack` (166)

7.28.2 Method overview

Page	Property	Description
165	AtLeast	Check whether the list contains a certain number of elements.
164	Count	Number of elements on the list.
164	Create	Create a new ordered list
164	Destroy	Free an ordered list
165	Peek	Return the next element to be popped from the list.
165	Pop	Remove an element from the list.
165	Push	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` (??)

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` (??)

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` (??)

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` (??)

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` (??), `TOrderedList.Peek` (??)

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` (??), `TOrderedList.Push` (??)

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` (??), without actually popping it from the list.

Errors: None.

See also: `TOrderedList.Pop` (??), `TOrderedList.Push` (??)

7.29 TQueue

7.29.1 Description

`TQueue` is a descendent of `TOrderedList` ([163](#)) which implements `Push` (??) and `Pop` (??) behaviour as a queue: what is first pushed on the queue, is popped of first (FIFO: First in, first out).

`TQueue` offers no new methods, it merely implements some abstract methods introduced by `TOrderedList` ([163](#))

See also: `TOrderedList` ([163](#)), `TObjectQueue` ([161](#)), `TStack` ([166](#))

7.30 TStack

7.30.1 Description

`TStack` is a descendent of `TOrderedList` ([163](#)) which implements `Push` (??) and `Pop` (??) behaviour as a stack: what is last pushed on the stack, is popped of first (LIFO: Last in, first out).

`TStack` offers no new methods, it merely implements some abstract methods introduced by `TOrderedList` ([163](#))

See also: `TOrderedList` ([163](#)), `TObjectStack` ([162](#)), `TQueue` ([166](#))

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` (168) 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` (??) to filter events that are sent to the system log.

`TExceptionHandler` = procedure (Sender: TObject; E: Exception) of object

`TExceptionHandler` is the prototype for the exception handling events in `TCustomApplication`.

8.3.2 Variables

`CustomApplication` : `TCustomApplication` = Nil

`CustomApplication` contains the global application instance. All descendents of `TCustomApplication` (168) 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` (??), `ShowException` (??), `OnException` (??) and `StopOnException` (??).
- Command-line parameter parsing in `FindOptionIndex` (??), `GetOptionValue` (??), `CheckOptions` (??) and `HasOption` (??)
- Environment variable handling in `GetEnvironmentList` (??) and `EnvironmentVariable` (??).

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

8.4.2 Method overview

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

8.4.3 Property overview

Page	Property	Access	Description
177	CaseSensitiveOptions	rw	Are options interpreted case sensitive or not
175	ConsoleApplication	r	Is the application a console application or not
176	EnvironmentVariable	r	Environment variable access
177	EventLogFilter	rw	Event to filter events, before they are sent to the system log
173	ExeName	r	Name of the executable.
174	HelpFile	rw	Location of the application help file.
175	Location	r	Application location
175	OnException	rw	Exception handling event
176	OptionChar	rw	Command-line switch character
176	ParamCount	r	Number of command-line parameters
175	Params	r	Command-line parameters
177	StopOnException	rw	Should the program loop stop on an exception
174	Terminated	r	Was <code>Terminate</code> called or not
174	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` (??)

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` (??)

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

See also: `ShowException (??)`, `StopOnException (??)`, `Terminated (??)`, `Run (??)`

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 (??)`, `TCustomApplication.Terminated (??)`

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 (??)`. If `TCustomApplication.StopOnException (??)` 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 (??)`, `TCustomApplication.StopException (??)`

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 (??)`, `TCustomApplication.StopException (??)`

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, descendent classes should in their `DoRun` method, check the value of the `Terminated (??)` property and properly shut down the application if it is set to `True`.

See also: `TCustomApplication.Terminated (??)`, `TCustomApplication.Run (??)`

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 (??)` property, the search is performed case sensitive or case insensitive.

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

See also: `HasOption (??)`, `GetOptionValue (??)`, `CheckOptions (??)`, `CaseSensitiveOptions (??)`, `OptionChar (??)`

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` (??) property, the search is performed case sensitive or case insensitive.

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

See also: `FindOptionIndex` (??), `HasOption` (??), `CheckOptions` (??), `CaseSensitiveOptions` (??), `OptionChar` (??)

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` (??) property, the search is performed case sensitive or case insensitive.

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

See also: `FindOptionIndex` (??), `GetOptionValue` (??), `CheckOptions` (??), `CaseSensitiveOptions` (??), `OptionChar` (??)

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: TStrings;AllErrors: Boolean)`
`: string`
`function CheckOptions(const ShortOptions: string;`
`const LongOpts: Array of ;AllErrors: Boolean)`
`: string`
`function CheckOptions(const ShortOptions: string;const LongOpts: string;`
`AllErrors: Boolean) : string`

Visibility: public

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

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

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

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

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

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

Errors: if an error was found during the check, the return value is a string describing the error.

See also: `FindOptionIndex` (??), `GetOptionValue` (??), `HasOption` (??), `CaseSensitiveOptions` (??), `OptionChar` (??)

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` (??)

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, descendent classes should override this method to provide appropriate logging: they should write the message `Msg` with type `EventType` to some log mechanism such as `#fcl.eventlog.TEventLog` (413)

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` (??)

8.4.19 TCustomApplication.Terminated

Synopsis: Was `Terminate` called or not

Declaration: `Property Terminated : Boolean`

Visibility: `public`

Access: `Read`

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

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` (??)

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 (??)`.

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

See also: `ShowEvent (??)`

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 (??)`

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 (??)`. It is equivalent to calling `ParamStr (??)`.

See also: `ParamCount (??)`, `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 (??)` property.

See also: `Params (??)`, `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 (??)` can be used.

See also: `GetEnvironmentList (??)`, `TCustomApplication.Params (??)`

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 (??)`, `GetOptionValue (??)`, `HasOption (??)`, `CaseSensitiveOptions (??)`, `CheckOptions (??)`

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 (??) and CheckOptions (??) 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 (??), GetOptionValue (??), HasOption (??), OptionChar (??), CheckOptions (??)

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

See also: Run (??), HandleException (??), Terminate (??)

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 (??) 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 (??)

Chapter 9

Reference for unit 'daemonapp'

9.1 Used units

Table 9.1: Used units by unit 'daemonapp'

Name	Page
Classes	??
CustApp	167
eventlog	411
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 descendents of this class can be implemented and instantiated in a single daemon application. For more information, see `TDaemon` ([194](#)).

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` ([199](#)).

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` ([202](#)).

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` ([413](#)) 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 (202) definition.

```
TCustomDaemonMapperClass = Class of TCustomDaemonMapper
```

TCustomDaemonMapperClass is the class of TCustomDaemonMapper. It is used in the RegisterDaemonMapper (184) 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 (185) 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
<code>drmInstall</code>	Daemon install mode (windows only)
<code>drmRun</code>	Daemon is running normally
<code>drmUninstall</code>	Daemon uninstall mode (windows only)
<code>drmUnknown</code>	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
<code>esCritical</code>	Error is logged, and startup is stopped if last known good configuration is active, or system is restarted using last known good configuration
<code>esIgnore</code>	Ignore startup errors
<code>esNormal</code>	Error is logged, but startup continues
<code>esSevere</code>	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
<code>stDevice</code>	Device driver
<code>stFileSystem</code>	File system driver
<code>stWin32</code>	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 (187) descendant. When the `Application` (183) 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 = ('Stopped', 'Start Pending', 'Stop

Names for various service statuses

`DefaultDaemonOptions` : TDaemonOptions = [doAllowStop, doAllowPause]

`DefaultDaemonOptions` are the default options with which a daemon definition (TDaemonDef (202)) is created.

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

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 (187) instance used by this application. The instance is created at the first invocation of this function, so it is possible to use `RegisterDaemonApplicationClass` (184) to register an alternative TCustomDaemonApplication class to run the application.

See also: TCustomDaemonApplication (187), RegisterDaemonApplicationClass (184)

9.5.2 DaemonError

Synopsis: Raise an EDaemon exception

Declaration: `procedure DaemonError(Msg: string)`
`procedure DaemonError(Fmt: string; Args: Array of const)`

Visibility: default

Description: `DemonError` raises an EDaemon (185) exception with message `Msg` or it formats the message using `Fmt` and `Args`.

See also: EDaemon (185)

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 (187) descendent which will be used when creating the global Application (183) instance. Only the last registered class pointer will be used.

See also: TCustomDaemonApplication (187), Application (183)

9.5.4 RegisterDaemonClass

Synopsis: Register daemon

Declaration: `procedure RegisterDaemonClass(AClass: TCustomDaemonClass)`

Visibility: default

Description: `RegisterDaemonClass` must be called for each TCustomDaemon (185) descendent that is used in the class: the class pointer and class name are used by the TCustomDaemonMapperClass (181) class to create a TCustomDaemon instance when a daemon is required.

See also: TCustomDaemonMapperClass (181), TCustomDaemon (185)

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 (187) code, so registering an alternative mapping class should happen in the initialization section of the application units.

See also: TCustomDaemonApplication (187), TCustomDaemonMapperClass (181)

9.6 EDaemon

9.6.1 Description

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

See also: DaemonError ([184](#))

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 ([194](#)) can be designed to implement the behaviour of the daemon.

See also: TDaemon ([194](#)), TDaemonDef ([202](#)), TDaemonController ([199](#)), TDaemonApplication ([199](#))

9.7.2 Method overview

Page	Property	Description
185	LogMessage	Log a message to the system log
186	ReportStatus	Report the current status to the operating system

9.7.3 Property overview

Page	Property	Access	Description
187	Controller	r	TDaemonController instance controlling this daemon instance
186	DaemonThread	r	Thread in which daemon is running
186	Definition	r	The definition used to instantiate this daemon instance
187	Logger	r	TEventLog instance used to send messages to the system log
187	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 ([413](#)) instance is used to actually send messages to the system log.

The message is sent with an 'error' flag (using TEventLog.Error ([416](#))).

Errors: None.

See also: ReportStatus ([??](#))

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` (??)

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` (202) 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` (202)

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 it's own thread, none of which are the main thread of the application. The application main thread is used to handle control messages coming from the operating system.

See also: `Controller` (??)

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 (??)

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 (??)

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

See also: LogMessage (??), TEventLog (413)

9.8 TCustomDaemonApplication

9.8.1 Description

TCustomDaemonApplication is a TCustomApplication (168) 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 (194) instances by checking the TCustomDaemonMapperClass (181) instance that contains the daemon maps.

See also: TCustomApplication (168), TCustomDaemonMapperClass (181)

9.8.2 Method overview

Page	Property	Description
188	CreateDaemon	Create daemon instance
190	CreateForm	Create a component
188	Destroy	Clean up the TCustomDaemonApplication instance
189	InstallDaemons	Install all daemons.
189	RunDaemons	Run all daemons.
188	ShowException	Show an exception
190	ShowHelp	Display a help message
189	StopDaemons	Stop all daemons
189	UnInstallDaemons	Uninstall all daemons

9.8.3 Property overview

Page	Property	Access	Description
191	EventLog	r	Event logger instance
191	GuiHandle	rw	Handle of GUI loop main application window handle
191	GUIMainLoop	rw	GUI main loop callback
190	OnRun	rw	Event executed when the daemon is run.
191	RunMode	r	Application mode

9.8.4 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` (??)

9.8.5 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.

9.8.6 TCustomDaemonApplication.CreateDaemon

Synopsis: Create daemon instance

Declaration: `function CreateDaemon(DaemonDef: TDaemonDef) : TCustomDaemon`

Visibility: `public`

Description: `CreateDaemon` is called whenever a `TCustomDaemon` ([185](#)) instance must be created from a `TDaemonDef` ([202](#)) daemon definition, passed in `DaemonDef`. It initializes the `TCustomDaemon` instance, and creates a controller instance of type `TDaemonController` ([199](#)) to control the daemon. Finally, it assigns the created daemon to the `TDaemonDef.Instance` (??) property.

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

See also: `TDaemonController` (199), `TCustomDaemon` (185), `TDaemonDef` (202), `TDaemonDef.Instance` (??)

9.8.7 `TCustomDaemonApplication.StopDaemons`

Synopsis: Stop all daemons

Declaration: `procedure StopDaemons(Force: Boolean)`

Visibility: public

Description: `StopDaemons` sends the `STOP` control code to all daemons, or the `SHUTDOWN` control code in case `Force` is `True`.

See also: `TDaemonController.Controller` (??), `TCustomDaemonApplication.UnInstallDaemons` (??), `TCustomDaemonApplication.RunDaemons` (??)

9.8.8 `TCustomDaemonApplication.InstallDaemons`

Synopsis: Install all daemons.

Declaration: `procedure InstallDaemons`

Visibility: public

Description: `InstallDaemons` installs all known daemons, i.e. registers them with the service manager on Windows. This method is called if the application is run with the `-i` or `-install` or `/install` command-line option.

See also: `TCustomDaemonApplication.UnInstallDaemons` (??), `TCustomDaemonApplication.RunDaemons` (??), `TCustomDaemonApplication.StopDaemons` (??)

9.8.9 `TCustomDaemonApplication.RunDaemons`

Synopsis: Run all daemons.

Declaration: `procedure RunDaemons`

Visibility: public

Description: `RunDaemons` runs (starts) all known daemons. This method is called if the application is run with the `-r` or `-run` methods.

See also: `TCustomDaemonApplication.UnInstallDaemons` (??), `TCustomDaemonApplication.InstallDaemons` (??), `TCustomDaemonApplication.StopDaemons` (??)

9.8.10 `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 (??)`, `TCustomDaemonApplication.InstallDaemons (??)`, `TCustomDaemonApplication.StopDaemons (??)`

9.8.11 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.

9.8.12 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 (??)`

9.8.13 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 (??)`, `TCustomDaemonApplication.InstallDaemons (??)`, `TCustomDaemonApplication.UnInstallDaemons (??)`

9.8.14 TCustomDaemonApplication.EventLog

Synopsis: Event logger instance

Declaration: `Property EventLog : TEventLog`

Visibility: public

Access: Read

Description: `EventLog` is the `TEventLog` (413) instance which is used to log events to the system log with the `Log (??)` method. It is created when the application instance is created, and destroyed when the application is destroyed.

See also: `TEventLog` (413), `Log (??)`

9.8.15 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 (??)`

9.8.16 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 (??)` procedure exists, and the application creates and runs a message loop by itself.

See also: `GUIMainLoop (??)`

9.8.17 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 (??)`, `RunDaemons (??)` or `UnInstallDaemons (??)` is called.

See also: `InstallDaemons (??)`, `RunDaemons (??)`, `UnInstallDaemons (??)`

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 (206) 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 (202) definitions and the actual TDaemon (194) instances that will be used to run the daemons.

The TCustomDaemonMapper class has no support for streaming. The TDaemonMapper (208) class has support for streaming (and hence visual designing).

See also: TDaemon (194), TDaemonDef (202), TDaemonDefs (206), TDaemonMapper (208)

9.9.2 Method overview

Page	Property	Description
192	Create	Create a new instance of TCustomDaemonMapper
192	Destroy	Clean up and destroy a TCustomDaemonMapper instance.

9.9.3 Property overview

Page	Property	Access	Description
193	DaemonDefs	rw	Collection of daemons
193	OnCreate	rw	Event called when the daemon mapper is created
193	OnDestroy	rw	Event called when the daemon mapper is freed.
194	OnInstall	rw	Event called when the daemons are installed
193	OnRun	rw	Event called when the daemons are executed.
194	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 (206) 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 (206), TCustomDaemonApplication (187), TCustomDaemonMapper.Destroy (??)

9.9.5 TCustomDaemonMapper.Destroy

Synopsis: Clean up and destroy a TCustomDaemonMapper instance.

Declaration: destructor Destroy; Override

Visibility: public

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

See also: [TDaemonDefs \(206\)](#), [TCustomDaemonMapper.Create \(??\)](#)

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 \(187\)](#)

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 \(??\)](#), [OnUnInstall \(??\)](#), [OnCreate \(??\)](#), [OnDestroy \(??\)](#)

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 (??)` event.

See also: [OnCreate \(??\)](#), [OnInstall \(??\)](#), [OnUnInstall \(??\)](#), [OnCreate \(??\)](#)

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` (??), `OnUnInstall` (??), `OnCreate` (??), `OnDestroy` (??)

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` (??), `OnUnInstall` (??), `OnCreate` (??), `OnDestroy` (??)

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` (??), `OnInstall` (??), `OnCreate` (??), `OnDestroy` (??)

9.10 TDaemon

9.10.1 Description

`TDaemon` is a `TCustomDaemon` ([185](#)) descendent which is meant for development in a visual environment: it contains event handlers for all major operations. Whenever a `TCustomDaemon` method is executed, it's execution is shunted to the event handler, which can be filled with code in the IDE.

All the events of the daemon are executed in the thread in which the daemon's controller is running (as given by `DaemonThread` (??)), which is not the main program thread.

See also: `TCustomDaemon` ([185](#)), `TDaemonController` ([199](#))

9.10.2 Property overview

Page	Property	Access	Description
198	AfterInstall	rw	Called after the daemon was installed
198	AfterUnInstall	rw	Called after the daemon is uninstalled
197	BeforeInstall	rw	Called before the daemon will be installed
198	BeforeUnInstall	rw	Called before the daemon is uninstalled
195	Definition		
196	OnContinue	rw	Daemon continue
198	OnControlCode	rw	Called when a control code is received for the daemon
197	OnExecute	rw	Daemon execute event
196	OnPause	rw	Daemon pause event
197	OnShutDown	rw	Daemon shutdown
195	OnStart	rw	Daemon start event
196	OnStop	rw	Daemon stop event
195	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 (??)` 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 (??)` event handler, so the event handler can return at once.

See also: `TDaemon.OnStop (??)`, `TDaemon.OnExecute (??)`, `TDaemon.OnContinue (??)`, `ReportStatus (??)`

9.10.6 TDaemon.OnStop

Synopsis: Daemon stop event

Declaration: `Property OnStop : TDaemonOKEvent`

Visibility: published

Access: Read,Write

Description: `OnStart` is the event called when the daemon must be stopped. This event handler should return as quickly as possible. If it must perform lengthy operations, it is best to report the status to the operating system at regular intervals using the `ReportStatus (??)` method.

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

See also: `TDaemon.OnStart (??)`, `TDaemon.OnPause (??)`, `ReportStatus (??)`

9.10.7 TDaemon.OnPause

Synopsis: Daemon pause event

Declaration: `Property OnPause : TDaemonOKEvent`

Visibility: published

Access: Read,Write

Description: `OnPause` is the event called when the daemon must be stopped. This event handler should return as quickly as possible. If it must perform lengthy operations, it is best to report the status to the operating system at regular intervals using the `ReportStatus (??)` method.

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

See also: `TDaemon.OnStop (??)`, `TDaemon.OnContinue (??)`, `ReportStatus (??)`

9.10.8 TDaemon.OnContinue

Synopsis: Daemon continue

Declaration: `Property OnContinue : TDaemonOKEvent`

Visibility: published

Access: Read,Write

Description: `OnPause` is the event called when the daemon must be stopped. This event handler should return as quickly as possible. If it must perform lengthy operations, it is best to report the status to the operating system at regular intervals using the `ReportStatus (??)` method.

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

See also: `TDaemon.OnStart (??)`, `TDaemon.OnPause (??)`, `ReportStatus (??)`

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` (??), this is the place where the thread's executed should be stopped or the thread freed from memory.

See also: `TDaemon.OnStart` (??), `TDaemon.OnPause` (??), `ReportStatus` (??)

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` (??), `TDaemon.OnStop` (??)

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` (??), `BeforeUnInstall` (??), `AfterUnInstall` (??)

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 succesfully installed.

See also: `BeforeInstall` (??), `BeforeUnInstall` (??), `AfterUnInstall` (??)

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` (??), `AfterInstall` (??), `AfterUnInstall` (??)

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 succesfully uninstalled.

See also: `BeforeInstall` (??), `AfterInstall` (??), `BeforeUnInstall` (??)

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` (??)

9.11 TDaemonApplication

9.11.1 Description

`TDaemonApplication` is the default `TCustomDaemonApplication` (187) descendent that is used to run the daemon application. It is possible to register an alternative `TCustomDaemonApplication` class (using `RegisterDaemonApplicationClass` (184)) to run the application in a different manner.

See also: `TCustomDaemonApplication` (187), `RegisterDaemonApplicationClass` (184)

9.12 TDaemonController

9.12.1 Description

`TDaemonController` is a class that is used by the `TDaemonApplication` (199) 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` (185), `TDaemonApplication` (199)

9.12.2 Method overview

Page	Property	Description
200	Controller	Controller
199	Create	Create a new instance of the <code>TDaemonController</code> class
200	Destroy	Free a <code>TDaemonController</code> instance.
200	Main	Daemon main entry point
201	ReportStatus	Report the status to the operating system.
200	StartService	Start the service

9.12.3 Property overview

Page	Property	Access	Description
202	CheckPoint		Send checkpoint signal to the operating system
201	Daemon	r	Daemon instance this controller controls.
201	LastStatus	r	Last reported status
201	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` (199) instance, and `AOwner` will be set to the global `TDaemonApplication` (199) instance.

See also: `TDaemonApplication` (199), `Destroy` (??)

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 (??)

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 (??)

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 ([209](#))

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 (??)

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 (??)`, `TDaemonController.LastStatus (??)`

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 : TStrings`

Visibility: `public`

Access: `Read`

Description: `Params` contains the parameters passed to the daemon application by the operating system, comparable to the application's command-line parameters. The property is set by the `Main (??)` 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 (??)`

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` (??)

9.13 TDaemonDef

9.13.1 Description

`TDaemonDef` contains the definition of a daemon in the application: The name of the daemon, which `TCustomDaemon` (185) descendent should be started to run the daemon, a description, and various other options should be set in this class. The global `TDaemonApplication` instance maintains a collection of `TDaemonDef` instances and will use these definitions to install or start the various daemons.

See also: `TDaemonApplication` (199), `TDaemon` (194)

9.13.2 Method overview

Page	Property	Description
202	Create	Create a new <code>TDaemonDef</code> instance
203	Destroy	Free a <code>TDaemonDef</code> from memory

9.13.3 Property overview

Page	Property	Access	Description
203	<code>DaemonClass</code>	r	<code>TDaemon</code> class to use for this daemon
203	<code>DaemonClassName</code>	rw	Name of the <code>TDaemon</code> class to use for this daemon
204	<code>Description</code>	rw	Description of the daemon
204	<code>DisplayName</code>	rw	Displayed name of the daemon (service)
205	<code>Enabled</code>	rw	Is the daemon enabled or not
203	<code>Instance</code>	rw	Instance of the daemon class
206	<code>LogStatusReport</code>	rw	Log the status report to the system log
204	<code>Name</code>	rw	Name of the daemon (service)
205	<code>OnCreateInstance</code>	rw	Event called when a daemon is instantiated
205	<code>Options</code>	rw	Service options
204	<code>RunArguments</code>	rw	Additional command-line arguments when running daemon.
205	<code>WinBindings</code>	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` ([206](#))

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 it's name in `DaemonClassName` ([??](#)).

See also: `DaemonClassName` ([??](#)), `TDaemonMapper` ([208](#))

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` ([194](#)) instance that is used when the service is in operation at runtime.

See also: `TDaemonDef.DaemonClass` ([??](#))

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 (??)` creates an instance of the daemon.

See also: `TDaemonDef.Instance (??)`, `TDaemonDef.DaemonClass (??)`, `RegisterDaemonClass` ([184](#))

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 (??)`

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 (??)`

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` ([213](#))

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` (??)

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` (192) instance will create a collection and maintain it.

See also: `TCustomDaemonMapper` (192), `TDaemonDef` (202)

9.14.2 Method overview

Page	Property	Description
206	Create	Create a new instance of a <code>TDaemonDefs</code> collection.
207	<code>DaemonDefByName</code>	Find and return instance of daemon definition with given name.
207	<code>FindDaemonDef</code>	Find and return instance of daemon definition with given name.
207	<code>IndexOfDaemonDef</code>	Return index of daemon definition

9.14.3 Property overview

Page	Property	Access	Description
207	<code>Daemons</code>	rw	Indexed access to <code>TDaemonDef</code> 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` (192) instance will create a collection and maintain it.

See also: `TDaemonDef` (202)

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` (202) instance with a name equal to `DemonName`, and returns it's index. It returns -1 if no definition was found with this name. The search is case insensitive.

See also: `TDaemonDefs.FindDaemonDef` (??), `TDaemonDefs.DaemonDefByName` (??)

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` (202) instance whose name matches `DemonName`. If no definition is found, `Nil` is returned.

See also: `TDaemonDefs.IndexOfDaemonDef` (??), `TDaemonDefs.DaemonDefByName` (??)

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` (202) instance whose name matches `DemonName`. If no definition is found, an `EDaemon` (185) exception is raised.

The `FindDaemonDef` (??) call does not raise an error, but returns `Nil` instead.

Errors: If no definition is found, an `EDaemon` (185) exception is raised.

See also: `TDaemonDefs.IndexOfDaemonDef` (??), `TDaemonDefs.FindDaemonDef` (??)

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` (202)

9.15 TDaemonMapper

9.15.1 Description

`TDaemonMapper` is a direct descendent of `TCustomDaemonMapper` (192), but introduces no new functionality. Its sole purpose is to make it possible for an IDE to stream the `TDaemonMapper` instance.

For this purpose, it overrides the `Create` constructor and tries to find a resource with the same name as the class name, and tries to stream the instance from this resource.

If the instance should not be streamed, the `CreateNew (??)` constructor can be used instead.

See also: `CreateNew (??)`, `Create (??)`

9.15.2 Method overview

Page	Property	Description
208	<code>Create</code>	Create a new <code>TDaemonMapper</code> instance and initializes it from streamed resources.
208	<code>CreateNew</code>	Create a new <code>TDaemonMapper</code> 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 (??)` 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 (??)`

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 (??)`

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 it's own thread.

It should not be necessary to create these threads manually, the TDaemonController (199) class will take care of this.

See also: TDaemonController (199), TDaemon (194)

9.16.2 Method overview

Page	Property	Description
210	CheckControlMessage	Check if a control message has arrived
210	ContinueDaemon	Continue the daemon
209	Create	Create a new thread
209	Execute	Run the daemon
211	InterrogateDaemon	Report the daemon status
210	PauseDaemon	Pause the daemon
211	ShutDownDaemon	Shut down daemon
210	StopDaemon	Stops the daemon

9.16.3 Property overview

Page	Property	Access	Description
211	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 (??)

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 (??)

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 (??)` method, and terminates the thread.

See also: `TDaemon.Stop (??)`, `TDaemonThread.PauseDaemon (??)`, `TDaemonThread.ShutDownDaemon (??)`

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 (??)` method, and suspends the thread. It returns `True` if the attempt was successful.

See also: `TDaemon.Pause (??)`, `TDaemonThread.StopDaemon (??)`, `TDaemonThread.ContinueDaemon (??)`, `TDaemonThread.ShutDownDaemon (??)`

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 (??)` method. It returns `True` if the attempt was successful.

See also: `TDaemon.Continue (??)`, `TDaemonThread.StopDaemon (??)`, `TDaemonThread.PauseDaemon (??)`, `TDaemonThread.ShutDownDaemon (??)`

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 (??)` function. The thread is terminated by this method.

See also: `TDaemon.Shutdown (??)`, `TDaemonThread.StopDaemon (??)`, `TDaemonThread.PauseDaemon (??)`, `TDaemonThread.ContinueDaemon (??)`

9.16.11 TDaemonThread.InterrogateDaemon

Synopsis: Report the daemon status

Declaration: `function InterrogateDaemon : Boolean; Virtual`

Visibility: public

Description: `InterrogateDaemon` simply calls `TDaemon.ReportStatus (??)` for the daemon that is running in this thread. It always returns `True`.

See also: `TDaemon.ReportStatus (??)`

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` ([194](#))

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` ([212](#)) instances.

See also: `TDependency` ([212](#))

9.17.2 Method overview

Page	Property	Description
212	Create	Create a new instance of a <code>TDependencies</code> collection.

9.17.3 Property overview

Page	Property	Access	Description
212	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` ([212](#)).

See also: `TDependency` ([212](#))

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` ([211](#)), `TDaemonDef` ([202](#))

9.18.2 Method overview

Page	Property	Description
213	Assign	Assign <code>TDependency</code> instance to another

9.18.3 Property overview

Page	Property	Access	Description
213	IsGroup	rw	Name refers to a service group
213	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 (??)

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 (??)

9.19 TWinBindings

9.19.1 Description

TWinBindings contains windows-specific properties for the daemon definition (in TDaemonDef.WinBindings (??)). If the daemon should not run on Windows, then the properties can be ignored.

See also: TDaemonDef (202), TDaemonDef.WinBindings (??)

9.19.2 Method overview

Page	Property	Description
214	Assign	Copies all properties
214	Create	Create a new TWinBindings instance
214	Destroy	Remove a TWinBindings instance from memory

9.19.3 Property overview

Page	Property	Access	Description
215	Dependencies	rw	Service dependencies
214	ErrCode	rw	Service specific error code
217	ErrorSeverity	rw	Error severity in case of startup failure
215	GroupName	rw	Service group name
217	IDTag	rw	Location in the service group
216	Password	rw	Password for service startup
217	ServiceType	rw	Type of service
216	StartType	rw	Service startup type.
216	UserName	rw	Username to run service as
216	WaitHint	rw	Timeout wait hint
215	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` ([202](#)), `TDaemonDef.WinBindings` (??), `TWinBindings.Dependencies` (??)

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` (??), `TWinBindings.Create` (??)

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 (??)` to the windows service manager. If it is zero, then the contents of `Win32ErrCode (??)` are reported. If it is nonzero, then the windows-errorcode is set to `ERROR_SERVICE_SPECIFIC_ERROR`.

See also: `TWinBindings.Win32ErrCode (??)`

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 (??)` to the windows service manager. It is sent if `ErrCode (??)` is zero.

See also: `ErrCode (??)`

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.IsGroup (??)`

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` (??)

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` (??) property.

This property is only used during installation of the service.

See also: `Password` (??)

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` (??)

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` (268) representing a set of records from a database, `TField` (316) which represents the contents of a field in a record, `TDataSource` (304) which acts as an event distributor on behalf of a dataset and `TParams` (388) which can be used to parametrize queries. The databases connections themselves are abstracted in the `TDatabase` (258) class.

10.3 Constants, types and variables

10.3.1 Constants

```
DefaultFieldClasses : Array[TFieldType] of TFieldClass = (Tfield, TStringField, TSma
```

`DefaultFieldClasses` contains the `TField` (316) descendent class to use when a `TDataset` instance needs to create fields based on the `TFieldDefs` (344) field definitions when opening the dataset. The entries can be set to create customized `TField` descendents for certain field datatypes in all datasets.

```
dsEditModes = [dsEdit, dsInsert, dsSetKey]
```

`dsEditModes` contains the various values of `TDataset.State` (??) 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` (??) for which data can be written to the dataset buffer.

```
FieldTypeNames : Array[TFieldType] of = ('Unknown', 'String', 'Smallint', 'Integer'
```

`FieldTypeNames` contains the names (in english) for the various field data types.

```
FieldTypetoVariantMap : Array[TFieldType] of Integer = (varError, varOleStr, varSma
```

`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` (268) or one of its descendents.

```
PBufferList = ^TBufferList
```

`PBufferList` is a pointer to a structure of type `TBufferList` (221). 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 (??) method. It is an opaque type, and should not be used any more, it is superseded by the TBookmarkStr (221) 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 (268) 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` (??) 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` (268) 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` (222) 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` (258) class.

`TDataChangeEvent = procedure(Sender: TObject; Field: TField) of object`

`TDataChangeEvent` is the event handler prototype for the `TDatasource.OnDataChange` (??) 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 (304) instances connected to a TDataSet (268) 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 (268) 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 (??), TDataSet.OnPostError (??) and TDataSet.OnDeleteError (??) 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 (221).

```
TDataSetNotifyEvent = procedure (DataSet: TDataSet) of object
```

TDataSetNotifyEvent is used in most of the TDataSet (268) 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 (268) 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 (310)

```
TDBTransactionClass = Class of TDBTransaction
```

TDBTransactionClass is the class pointer for the TDBTransaction (311) 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 (??) property of TFieldDef (340).

TFieldAttributes = Set of TFieldAttribute

TFieldAttributes is used in the TFieldDef.Attributes (??) property to denote additional attributes of the underlying field.

TFieldChars = Set of Char

TFieldChars is a type used in the TField.ValidChars (??) 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 (??) 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 (224) 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 (316) class. It's 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 data (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 (316) underlying data, in the DataType (??) 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 (227) set is used in the TDataset.FilterOptions (??) 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 (??) property. The available values are described in the TFilterOption (226) type.

TFilterRecordEvent = procedure (DataSet: TDataSet; var Accept: Boolean) of object

TFilterRecordEvent is the prototype for the TDataset.OnFilterRecord (??) 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 (268) 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 descendents of TDataset (268) 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 (228) set type to describe all properties of an index definition as in TIndexDef (358).

```
TIndexOptions = Set of TIndexOption
```

TIndexOptions contains the set of properties that an index can have. It is used in the TIndexDef.Options (??) property to describe all properties of an index definition as in TIndexDef (358).

```
TLocateOption = (loCaseInsensitive, loPartialKey)
```

Table 10.14: Enumeration values for type TLocateOption

Value	Explanation
loCaseInsensitive	Perform a case-insensitive search
loPartialKey	Accept partial key matches

TLocateOption is used in the TDataset.Locate (??) call to enumerate the possible options available when locating a record in the dataset.

```
TLocateOptions = Set of TLocateOption
```

TLocateOptions is used in the TDataset.Locate (??) 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 the TCustomConnection.OnLogin (??) 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
psInterbase	Parameters are specified by a ? character
psPostgreSQL	Parameters are specified by a \$N character.
psSimulated	Parameters are specified by a \$N character.

`TParamStyle` denotes the style in which parameters are specified in a query. It is used in the `TParams.ParseSQL` (??) 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
ptInput	Input parameter
ptInputOutput	Input/output parameter
ptOutput	Output parameter, filled on result
ptResult	Result parameter
ptUnknown	Unknown type

`TParamType` indicates the kind of parameter represented by a `TParam` (376) 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 (229) 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 (268) has one or more of these flags.

TProviderFlags = Set of TProviderFlag

TProviderFlags is used for the TField.ProviderFlags (??) 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	
ctDelete	
ctInsert	
ctQuery	
ctSelect	
ctStoredProc	
ctTable	
ctUnknown	
ctUpdate	

TRecordBuffer = PAnsiChar

TRecordBufferBaseType = AnsiChar

TResolverResponse = (rrSkip, rrAbort, rrMerge, rrApply, rrIgnore)

Table 10.19: Enumeration values for type TResolverResponse

Value	Explanation
rrAbort	Abor 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 (268) navigation and data manipulation methods such as the TDataset.Refresh (??) 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 (268) class, but is defined on behalf of TDataset descendents that implement caching of updates: It indicates what should be done when the (delayed) applying of the updates fails. This event occurs long after the actual post or delete operation.

TUpdateKind = (ukModify,ukInsert,ukDelete)

Table 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 (232) 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` (235) exception, passing it `Msg`. If `Comp` is specified, the name of the component is prepended to the message.

See also: `DatabaseErrorFmt` (233), `EDatabaseError` (235)

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` (235) 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` (233), `EDatabaseError` (235)

10.4.4 DateTimeRecToDateTime

Synopsis: Convert `TDateTimeRec` record to a `TDateTime` value.

Declaration: `function DateTimeRecToDateTime(DT: TFieldType;Data: TDateTimeRec)`
`: TDateTime`

Visibility: default

Description: `DateTimeRecToDateTime` examines `Data` and `Dt` and uses `dt` to convert the timestamp in `Data` to a `TDateTime` value.

See also: `TFieldType` (225), `TDateTimeRec` (223), `DateTimeToDateTimeRec` (234)

10.4.5 DateTimeToDateTimeRec

Synopsis: Convert `TDateTime` value to a `TDateTimeRec` record.

Declaration: `function DateTimeToDateTimeRec (DT: TFieldType; Data: TDateTime)
: TDateTimeRec`

Visibility: default

Description: `DateTimeToDateTimeRec` examines `Data` and `Dt` and uses `dt` to convert the date/time value in `Data` to a `TDateTimeRec` record.

See also: `TFieldType` (225), `TDateTimeRec` (223), `DateTimeRecToDateTime` (233)

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` (??)

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` (??)

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` (233) call.

See also: `DatabaseError` (233), `DatabaseErrorFmt` (233)

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` (235)

10.6.2 Method overview

Page	Property	Description
235	Create	Create a new <code>EUpdateError</code> instance
236	Destroy	Free the <code>EupdateError</code> instance

10.6.3 Property overview

Page	Property	Access	Description
236	Context	r	Context in which exception occurred.
236	ErrorCode	r	Numerical error code.
236	OriginalException	r	
237	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` (235)

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` (??)

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` (??), `EUpdateError.ErrorCode` (??), `EUpdateError.PreviousError` (??)

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` (??), `EUpdateError.Context` (??), `EUpdateError.PreviousError` (??)

10.6.8 EUpdateError.OriginalException

Declaration: `Property OriginalException : Exception`

Visibility: `public`

Access: `Read`

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 (??)`, `EUpdateError.Context (??)`, `EUpdateError.OriginalException (??)`

10.7 IProviderSupport

10.7.1 Method overview

Page	Property	Description
237	<code>PSEndTransaction</code>	
237	<code>PSExecute</code>	
238	<code>PSExecuteStatement</code>	
238	<code>PSGetAttributes</code>	
238	<code>PSGetCommandText</code>	
238	<code>PSGetCommandType</code>	
238	<code>PSGetDefaultOrder</code>	
238	<code>PSGetIndexDefs</code>	
238	<code>PSGetKeyFields</code>	
238	<code>PSGetParams</code>	
238	<code>PSGetQuoteChar</code>	
239	<code>PSGetTableName</code>	
239	<code>PSGetUpdateException</code>	
239	<code>PSInTransaction</code>	
239	<code>PSIsSQLBased</code>	
239	<code>PSIsSQLSupported</code>	
239	<code>PSReset</code>	
239	<code>PSSetCommandText</code>	
239	<code>PSSetParams</code>	
239	<code>PSStartTransaction</code>	
240	<code>PSUpdateRecord</code>	

10.7.2 IProviderSupport.PSEndTransaction

Declaration: `procedure PSEndTransaction(ACommit: Boolean)`

Visibility: default

10.7.3 IProviderSupport.PSExecute

Declaration: `procedure PSExecute`

Visibility: default

10.7.4 IProviderSupport.PSExecuteStatement

Declaration: `function PSExecuteStatement(const ASQL: string; AParams: TParams;
ResultSet: Pointer) : Integer`

Visibility: default

10.7.5 IProviderSupport.PSGetAttributes

Declaration: `procedure PSGetAttributes(List: TList)`

Visibility: default

10.7.6 IProviderSupport.PSGetCommandText

Declaration: `function PSGetCommandText : string`

Visibility: default

10.7.7 IProviderSupport.PSGetCommandType

Declaration: `function PSGetCommandType : TPSCommandType`

Visibility: default

10.7.8 IProviderSupport.PSGetDefaultOrder

Declaration: `function PSGetDefaultOrder : TIndexDef`

Visibility: default

10.7.9 IProviderSupport.PSGetIndexDefs

Declaration: `function PSGetIndexDefs(IndexTypes: TIndexOptions) : TIndexDefs`

Visibility: default

10.7.10 IProviderSupport.PSGetKeyFields

Declaration: `function PSGetKeyFields : string`

Visibility: default

10.7.11 IProviderSupport.PSGetParams

Declaration: `function PSGetParams : TParams`

Visibility: default

10.7.12 IProviderSupport.PSGetQuoteChar

Declaration: `function PSGetQuoteChar : string`

Visibility: default

10.7.13 IProviderSupport.PSGetTableName

Declaration: `function PSGetTableName : string`

Visibility: default

10.7.14 IProviderSupport.PSGetUpdateException

Declaration: `function PSGetUpdateException(E: Exception; Prev: EUpdateError)
: EUpdateError`

Visibility: default

10.7.15 IProviderSupport.PSInTransaction

Declaration: `function PSInTransaction : Boolean`

Visibility: default

10.7.16 IProviderSupport.PSIsSQLBased

Declaration: `function PSIsSQLBased : Boolean`

Visibility: default

10.7.17 IProviderSupport.PSIsSQLSupported

Declaration: `function PSIsSQLSupported : Boolean`

Visibility: default

10.7.18 IProviderSupport.PSReset

Declaration: `procedure PSReset`

Visibility: default

10.7.19 IProviderSupport.PSSetCommandText

Declaration: `procedure PSSetCommandText(const CommandText: string)`

Visibility: default

10.7.20 IProviderSupport.PSSetParams

Declaration: `procedure PSSetParams(AParams: TParams)`

Visibility: default

10.7.21 IProviderSupport.PSStartTransaction

Declaration: `procedure PSStartTransaction`

Visibility: default

10.7.22 IProviderSupport.PSUpdateRecord

Declaration: `function PSUpdateRecord(UpdateKind: TUpdateKind; Delta: TDataSet)
: Boolean`

Visibility: default

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` ([316](#))

10.8.2 Method overview

Page	Property	Description
240	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` ([316](#)) class' fields.

See also: `TField` ([316](#))

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` (??) equals `ftBCD`). It initializes some of the properties of the `TField` ([316](#)) 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` ([355](#)) 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` ([268](#)), `TField` ([316](#)), `TFMTBCDField` ([355](#))

10.9.2 Method overview

Page	Property	Description
241	<code>CheckRange</code>	Check whether a values falls within the allowed range
241	<code>Create</code>	Create a new instance of a <code>TBCDField</code> class.

10.9.3 Property overview

Page	Property	Access	Description
242	<code>Currency</code>	rw	Does the field represent a currency amount
242	<code>MaxValue</code>	rw	Maximum value for the field
243	<code>MinValue</code>	rw	Minimum value for the field
242	<code>Precision</code>	rw	Precision of the BCD field
243	<code>Size</code>		Number of decimals after the decimal separator
241	<code>Value</code>	rw	Value of the field contents as a <code>Currency</code> 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` ([316](#)) properties to configure the instance for working with BCD data values.

See also: `TField` ([316](#))

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` (??) and `MaxValue` (??) properties. If the value lies outside of the allowed range, then `False` is returned.

See also: `MaxValue` (??), `MinValue` (??)

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` (??) property to a `currency` type field. It returns the same value as the `TField.AsCurrency` (??) field.

See also: `TField.Value` (??), `TField.AsCurrency` (??)

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` (??), 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` (??), `TBCDField.Value` (??)

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` (??) and `TField.Text` (??) 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` (??), `TField.Text` (??)

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` (??) should also be set, because it will also be checked.

See also: `TBCDField.MinValue` (??), `TBCDField.CheckRange` (??)

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` (??) should also be set, because it will also be checked.

See also: `TBCDField.MaxValue` (??), `TBCDField.CheckRange` (??)

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` (??) field.

See also: `TBCDField.Precision` (??)

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` (316) 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` (250) or `TVarBytesField` (397) should be created.

See also: `TDataset` (268), `TField` (316), `TBytesField` (250), `TVarBytesField` (397)

10.10.2 Method overview

Page	Property	Description
244	Create	Create a new instance of a <code>TBinaryField</code> class.

10.10.3 Property overview

Page	Property	Access	Description
244	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` ([316](#))

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` (??)

10.11 TBlobField

10.11.1 Description

`TBlobField` is the class used when a dataset must manage BLOB data. (`TField.DataType` (??) equals `ftBLOB`). It initializes some of the properties of the `TField` ([316](#)) 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` ([372](#)), `TWideMemoField` ([398](#)) or `TGraphicField` ([357](#))

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` ([268](#)), `TField` ([316](#)), `TMemoField` ([372](#)), `TWideMemoField` ([398](#)), `TGraphicField` ([357](#))

10.11.2 Method overview

Page	Property	Description
245	<code>Clear</code>	Clear the BLOB field's contents
245	<code>Create</code>	Create a new instance of a <code>TBlobField</code> class.
245	<code>IsBlob</code>	Is the field a blob field
245	<code>LoadFromFile</code>	Load the contents of the field from a file
246	<code>LoadFromStream</code>	Load the field's contents from stream
246	<code>SaveToFile</code>	Save field contents to a file
246	<code>SaveToStream</code>	Save the field's contents to stream
247	<code>SetFieldType</code>	Set field type

10.11.3 Property overview

Page	Property	Access	Description
247	BlobSize	r	Size of the current blob
248	BlobType	rw	Type of blob
247	Modified	rw	Has the field's contents been modified.
248	Size		Size of the blob field
248	Transliterate	rw	Should the contents of the field be transliterated
247	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 (316) properties to configure the instance for working with BLOB data.

See also: TField (316)

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 (??). It creates and immediately releases an empty blob stream in write mode, effectively clearing the contents of the BLOB field.

See also: TField.Clear (??), TField.IsNull (??)

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 (??)

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, en then calls `LoadFromStream (??)` to read the contents of the blob field from the file. The file is opened in read-only mode.

Errors: If the file does not exist or is nor available for reading, an exception will be raised.

See also: `LoadFromStream (??)`, `SaveToFile (??)`

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 (??)`, `LoadFromFile (??)`

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 (??)` 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 (??)`, `SaveToStream (??)`

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 (??)`, `LoadFromStream (??)`

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` (??)

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` (??), `TField.DataSize` (??)

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` (??)

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` (??) property.

See also: `TField.Value` (??), `TField.AsString` (??)

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` (??) 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` (??), `TDataset.Translate` (??)

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` (??), but with a restricted set of values. Setting `BlobType` is equivalent to setting the `TField.DataType` (??) property.

See also: `TField.DataType` (??)

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` (??) property instead.

See also: `BlobSize` (??)

10.12 TBooleanField

10.12.1 Description

`TBooleanField` is the field class used by `TDataset` (268) whenever it needs to manage boolean data (`TField.DataType` (??) equals `ftBoolean`). It overrides some properties and methods of `TField` (316) 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 (268), TField (316)

10.12.2 Method overview

Page	Property	Description
249	Create	Create a new instance of the TBooleanField class.

10.12.3 Property overview

Page	Property	Access	Description
249	DisplayValues	rw	Textual representation of the true and false values
249	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 (316) properties to configure it for working with boolean values.

See also: TField (316)

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 (??) by TBooleanField as a boolean value. It returns the same value as the TField.AsBoolean (??) property.

See also: TField.AsBoolean (??), TField.Value (??)

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` (??), `TField.Text` (??)

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` (??) equals `ftBytes`). It initializes some of the properties of the `TField` (316) 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` (268), `TField` (316), `TVarBytesField` (397)

10.13.2 Method overview

Page	Property	Description
250	<code>Create</code>	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` (316) properties to configure the instance for working with binary data values.

See also: `TField` (316)

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` (??) property, or a constraint enforced by the user on a particular dataset instance stored in `TCheckConstraint.CustomConstraint` (??)

See also: `TCheckConstraints` (252), `TCheckConstraint.ImportedConstraint` (??), `TCheckConstraint.CustomConstraint` (??)

10.14.2 Method overview

Page	Property	Description
251	Assign	Assign one constraint to another

10.14.3 Property overview

Page	Property	Access	Description
251	CustomConstraint	rw	User-defined constraint
251	ErrorMessage	rw	Message to display when the constraint is violated
252	FromDictionary	rw	True if the constraint is imported from a datadictionary
252	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 (??), TCheckConstraint.CustomConstraint (??)

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 ([268](#)) 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 (??)

See also: TCheckConstraint.ErrorMessage (??)

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 (??)

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` (268) descendents to indicate the source of the constraint, but is otherwise ignored.

See also: `TCheckConstraint.ImportedConstraint` (??)

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` (268) descendent.

See also: `TCheckConstraint.CustomConstraint` (??)

10.15 TCheckConstraints

10.15.1 Description

`TCheckConstraints` is a `TCollection` descendent which keeps a collection of `TCheckConstraint` (250) items. It overrides the `Add` (??) method to return a `TCheckConstraint` instance.

See also: `TCheckConstraint` (250)

10.15.2 Method overview

Page	Property	Description
253	Add	Add new <code>TCheckConstraint</code> item to the collection
253	Create	Create a new instance of the <code>TCheckConstraints</code> class.

10.15.3 Property overview

Page	Property	Access	Description
253	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` (268) 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` (250) class pointer.

See also: `TCheckConstraint` (250), `TDataset` (268)

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` (250) instance to the collection. it returns the newly added instance.

See also: `TCheckConstraint` (250), `#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` (268) when it needs to manage currency-valued data. (`TField.Datatype` (??) equals `ftCurrency`). It simply sets some `Tfield` (316) 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` (316), `TDataset` (268)

10.16.2 Method overview

Page	Property	Description
254	Create	Create a new instance of a <code>TCurrencyField</code> .

10.16.3 Property overview

Page	Property	Access	Description
254	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 (??)`) to be able to work with currency data.

See also: `TField` ([316](#)), `TCurrencyField.Currency (??)`

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 (??)` 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 (??)`

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 (??)`) and to deactivate the connection (`TCustomConnection.Close (??)`), plus a property to inspect the state (`Connected (??)`) of the connected.

See also: `TCustomConnection.Open (??)`, `TCustomConnection.Close (??)`, `TCustomConnection.Connected (??)`

10.17.2 Method overview

Page	Property	Description
255	Close	Close the connection
255	Destroy	Remove the <code>TCustomconnection</code> instance from memory
255	Open	Makes the connection to the server

10.17.3 Property overview

Page	Property	Access	Description
257	AfterConnect	rw	Event triggered after a connection is made.
257	AfterDisconnect	rw	Event triggered after a connection is closed
257	BeforeConnect	rw	Event triggered before a connection is made.
258	BeforeDisconnect	rw	Event triggered before a connection is closed
256	Connected	rw	Is the connection established or not
256	DataSetCount	r	Number of datasets connected to this connection
256	DataSets	r	Datasets linked to this connection
257	LoginPrompt	rw	Should the OnLogin be triggered
258	OnLogin	rw	Event triggered when a login prompt is shown.

10.17.4 TCustomConnection.Close

Synopsis: Close the connection

Declaration: `procedure Close`

Visibility: `public`

Description: `Close` closes the connection with the server if it was connected. Calling this method first triggers the `BeforeDisconnect` (??) 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` (??) event is triggered.

Calling the `Close` method is equivalent to setting the `Connected` (??) property to `False`.

Errors: If the connection cannot be broken for some reason, an `EDatabaseError` ([235](#)) exception will be raised.

See also: `TCustomConnection.BeforeDisconnect` (??), `TCustomConnection.AfterDisconnect` (??), `TCustomConnection.Open` (??), `TCustomConnection.Connected` (??)

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` (??)

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 (??)` event. If an exception is raised during the execution of that event handler, the connect process is aborted. If `LoginPrompt (??)` is `True`, the `OnLogin (??)` event handler is called. Only after this event, the connection is actually established. After the connection was established, the `AfterConnect (??)` event is triggered.

Calling the `Open` method is equivalent to setting the `Connected (??)` 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 (??)`, `TCustomConnection.LoginPrompt (??)`, `TCustomConnection.OnLogin (??)`, `TCustomConnection.AfterConnect (??)`, `TCustomConnection.Connected (??)`

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 (??)` array property. As implemented in `TCustomConnection`, this property is always zero. Descendent classes implement the actual count.

See also: `TDataset (268)`, `TCustomConnection.Datasets (??)`

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 (??)`.

See also: `DataSetCount (??)`

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 (??)`), or to `False` to break it (equivalent to calling `TCustomConnection.Close (??)`).

See also: `TCustomConnection.Open (??)`, `TCustomConnection.Close (??)`

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 (??)

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 (??). It can be used to open datasets, or indicate a connection status change.

See also: TCustomConnection.Open (??), TCustomConnection.BeforeConnect (??), TCustomConnection.OnLogin (??)

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 (??). It can be used for instance to indicate a connection status change.

See also: TCustomConnection.Close (??), TCustomConnection.BeforeDisconnect (??)

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 (??). 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 (??), TCustomConnection.AfterConnect (??), TCustomConnection.OnLogin (??)

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` (??).

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` (??), `TCustomConnection.AfterDisconnect` (??)

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` (??) property is `True`, after the `TCustomConnection.BeforeConnect` (??) event, but before the connection is actually established.

See also: `TCustomConnection.BeforeConnect` (??), `TCustomConnection.LoginPrompt` (??), `TCustomConnection.Open` (??)

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` (268), `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` (??) 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` (268), `TDatabase` (258)

10.18.2 Method overview

Page	Property	Description
259	<code>CloseDataSets</code>	Close all connected datasets
260	<code>CloseTransactions</code>	End all transactions
259	<code>Create</code>	Initialize a new <code>TDatabase</code> class instance.
259	<code>Destroy</code>	Remove a <code>TDatabase</code> instance from memory.
260	<code>EndTransaction</code>	End an active transaction.
260	<code>StartTransaction</code>	Start a new transaction.

10.18.3 Property overview

Page	Property	Access	Description
262	<code>Connected</code>	rw	Is the database connected
262	<code>DatabaseName</code>	rw	Database name or path
261	<code>Directory</code>	rw	Directory for the database
261	<code>IsSQLBased</code>	r	Is the database SQL based.
262	<code>KeepConnection</code>	rw	Should the connection be kept active
262	<code>Params</code>	rw	Connection parameters
260	<code>TransactionCount</code>	r	Number of transaction components connected to this database.
261	<code>Transactions</code>	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` ([310](#)), `TDBTransaction` ([311](#)), `TDatabase.Destroy` ([??](#))

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` ([??](#))

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` (??), `TDatabase.CloseTransactions` (??)

10.18.7 TDatabase.CloseTransactions

Synopsis: End all transactions

Declaration: `procedure CloseTransactions`

Visibility: public

Description: `CloseTransaction` calls `TDBTransaction.EndTransaction` (??) on all connected transactions. It is called automatically when the connection is closed, after all datasets are closed.

See also: `TCustomConnection.Close` (??), `TDatabase.CloseDatasets` (??)

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 should use a `TDBTransaction` (311) component instead and invoke the `TDBTransaction.StartTransaction` (??) method.

See also: `TDBTransaction` (311), `TDBTransaction.StartTransaction` (??)

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 should use a `TDBTransaction` (311) component instead and invoke the `TDBTransaction.EndTransaction` (??) method.

See also: `TDBTransaction` (311), `TDBTransaction.EndTransaction` (??)

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` (??) array property.

See also: `TDatabase.Transactions` (??)

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` (??)

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` (258).

See also: `TDatabase.Params` (??), `TDatabase.IsSQLBased` (??)

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` (??), `TDatabase.Directory` (??)

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` (??).

See also: `TCustomConnection.Connected` (??)

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` (??) 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` (??), `TDatabase.Params` (??)

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` (??)

10.18.17 TDatabase.Params

Synopsis: Connection parameters

Declaration: `Property Params : TStrings`

Visibility: published

Access: Read,Write

Description: `Params` is a catch-all storage mechanism for database connection parameters. It is a list of strings in the form of `Name=Value` pairs. Which name/value pairs are supported depends on the `TDatabase` descendent, but the `user_name` and `password` parameters are commonly used to store the login credentials for the database.

See also: `TDatabase.Directory` (??), `TDatabase.DatabaseName` (??)

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 (304) instance. It is a class that exists for component programmers, application coders should never need to use TDataLink or one of its descendents.

DB-Aware Component coders must use a TDataLink instance to handle all communication with a TDataSet (268) 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 (268), TDataSource (304), TDetailDataLink (315), TMasterDataLink (369)

10.19.2 Method overview

Page	Property	Description
263	Create	Initialize a new instance of TDataLink
264	Destroy	Remove an instance of TDataLink from memory
264	Edit	Set the dataset in edit mode, if possible
264	ExecuteAction	Execute action
265	UpdateAction	Update handler for actions
264	UpdateRecord	Called when the data in the dataset must be updated

10.19.3 Property overview

Page	Property	Access	Description
265	Active	r	Is the link active
265	ActiveRecord	rw	Currently active record
265	BOF	r	Is the dataset at the first record
266	BufferCount	rw	Set to the number of record buffers this datalink needs.
266	DataSet	r	Dataset this datalink is connected to
266	DataSource	rw	Datasource this datalink is connected to
266	DataSourceFixed	rw	Can the datasource be changed
267	Editing	r	Is the dataset in edit mode
267	Eof	r	
267	ReadOnly	rw	Is the link readonly
267	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 (??)

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 (??)

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 (??)` method of the `DataSource (??)` to which the datalink instance is coupled. If the `TDataSource.AutoEdit (??)` 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 (304), TDataLink.DataSource (??), TDataSource.Edit (??), TDataSource.AutoEdit (??)

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 (??)` notification (which could result from writing data to the dataset) will be blocked.

See also: TDataLink.RecordChange (??)

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 (??)

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` (??)

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` (??) whenever the `DataSource` (??) property is set.

See also: `TDataLink.DataSource` (??), `TDataLink.ReadOnly` (??), `TDataset.Active` (??)

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` (??), `TDataLink.EOF` (??)

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` (??) if the dataset is available, `True` otherwise.

See also: `TDataLink.EOF` (??), `TDataset.BOF` (??)

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` (??), `TDatalink.ActiveRecord` (??)

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` (??), `TDataset` (268)

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` (304) instance to get access to the dataset it is connected to. A datalink never points directly to a `TDataset` (268) 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` (268), `TDataSource` (304)

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` (??) 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` (??)

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 (??)` 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 (??)`, `TDataset.Edit (??)`

10.19.18 TDataLink.Eof

Synopsis:

Declaration: `Property Eof : Boolean`

Visibility: `public`

Access: `Read`

Description: `EOF` returns `TDataset.EOF (??)` if the dataset is available, `True` otherwise.

See also: `TDatalink.BOF (??)`, `TDataset.EOF (??)`

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 (??)` 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 (??)` property can be used to disable delivery of events to the datalink.

See also: `TDatalink.Active (??)`, `TDatalink.edit (??)`

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 (??)` property: `RecordCount` is always less than `Buffercount`.

See also: `TDatalink.BufferCount (??)`

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. Therefore `TDataSet` is never used directly, one always instantiates a descendent class.

Initially, no data is available: the dataset is inactive. The `Open (??)` 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 (??)` property). Likewise, the `Close (??)` 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 (??)`, `Prev (??)`, `First (??)` and `Last (??)` methods, it is possible to navigate through the records. The `EOF (??)` property will be `True` if the last row has been reached. Likewise, the `BOF (??)` 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 (??)` will be zero.

The `Append (??)` and `Insert (??)` methods can be used to insert new records to the set of records. The `TDataSet.Delete (??)` statement is used to delete the current record, and the `Edit (??)` 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 (??)` method, or can be undone using the `Cancel (??)` method.

The data in the various fields properties is available through the `Fields (??)` 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 \(316\)](#)

10.20.2 Method overview

Page	Property	Description
272	ActiveBuffer	Currently active memory buffer
273	Append	Append a new record to the data
273	AppendRecord	Append a new record to the dataset and fill with data
274	BookmarkValid	Test whether ABookMark is a valid bookmark.
274	Cancel	Cancel the current editing operation
274	CheckBrowseMode	Check whether the dataset is in browse mode.
274	ClearFields	Clear the values of all fields
275	Close	Close the dataset
275	CompareBookmarks	Compare two bookmarks
275	ControlsDisabled	Check whether the controls are disabled
272	Create	Create a new TDataset instance
276	CreateBlobStream	Create blob stream
276	CursorPosChanged	Indicate a change in cursor position
276	DataConvert	Convert data from/to native format
276	Delete	Delete the current record.
272	Destroy	Free a TDataset instance
277	DisableControls	Disable event propagation of controls
277	Edit	Set the dataset in editing mode.
278	EnableControls	Enable event propagation of controls
278	FieldByName	Search a field by name
278	FindField	Find a field by name
279	FindFirst	Find the first active record (deprecated)
279	FindLast	Find the last active record (deprecated)
279	FindNext	Find the next active record (deprecated)
279	FindPrior	Find the previous active record (deprecated)
280	First	Position the dataset on the first record.
280	FreeBookmark	Free a bookmark obtained with GetBookmark (deprecated)
280	GetBookmark	Get a bookmark pointer (deprecated)
281	GetCurrentRecord	Copy the data for the current record in a memory buffer
272	GetFieldData	Get the data for a field
281	GetFieldList	Return field instances in a list
281	GetFieldNames	Return a list of all available field names
281	GotoBookmark	Jump to bookmark
282	Insert	Insert a new record at the current position.
282	InsertRecord	Insert a new record with given values.
282	IsEmpty	Check if the dataset contains no data
282	IsLinkedTo	Check whether a datasource is linked to the dataset
283	IsSequenced	Is the data sequenced
283	Last	Navigate forward to the last record
283	Locate	Locate a record based on some key values
284	Lookup	Search for a record and return matching values.
284	MoveBy	Move the cursor position
284	Next	Go to the next record in the dataset.
285	Open	Activate the dataset: Fetch data into memory.
285	Post	Post pending edits to the database.
286	Prior	Go to the previous record
286	Refresh	Refresh the records in the dataset
286	Resync	Resynchronize the data buffer
273	SetFieldData	Store the data for a field
286	SetFields	Set a number of field values at once
287	Translate	Transliterate a buffer
287	UpdateCursorPos	Update cursor position
287	UpdateRecord	Indicate that the record contents have changed
288	UpdateStatus	Get the update status for the current record

10.20.3 Property overview

Page	Property	Access	Description
295	Active	rw	Is the dataset open or closed.
300	AfterCancel	rw	Event triggered after a Cancel operation.
297	AfterClose	rw	Event triggered after the dataset is closed
300	AfterDelete	rw	
298	AfterEdit	rw	Event triggered after the dataset is put in edit mode.
298	AfterInsert	rw	Event triggered after the dataset is put in insert mode.
297	AfterOpen	rw	Event triggered after the dataset is opened.
299	AfterPost	rw	Event called after changes have been posted to the underlying database
301	AfterRefresh	rw	Event triggered after the data has been refreshed.
301	AfterScroll	rw	Event triggered after the cursor has changed position.
296	AutoCalcFields	rw	How often should the value of calculated fields be calculated
299	BeforeCancel	rw	Event triggered before a Cancel operation.
297	BeforeClose	rw	Event triggered before the dataset is closed.
300	BeforeDelete	rw	Event triggered before a Delete operation.
298	BeforeEdit	rw	Event triggered before the dataset is put in edit mode.
297	BeforeInsert	rw	Event triggered before the dataset is put in insert mode.
296	BeforeOpen	rw	Event triggered before the dataset is opened.
299	BeforePost	rw	Event called before changes are posted to the underlying database
301	BeforeRefresh	rw	Event triggered before the data is refreshed.
300	BeforeScroll	rw	Event triggered before the cursor changes position.
288	BlockReadSize	rw	
288	BOF	r	Is the cursor at the beginning of the data (on the first record)
288	Bookmark	rw	Get or set the current cursor position
289	CanModify	r	Can the data in the dataset be modified
289	DataSource	r	Datasource this dataset is connected to.
290	DefaultFields	r	Is the dataset using persistent fields or not.
290	EOF	r	Indicates whether the last record has been reached.
291	FieldCount	r	Number of fields
291	FieldDefs	rw	Definitions of available fields in the underlying database
294	Fields	r	Indexed access to the fields of the dataset.
294	FieldValues	rw	Access to field values based on the field names.
294	Filter	rw	Filter to apply to the data in memory.
295	Filtered	rw	Is the filter active or not.
295	FilterOptions	rw	Options to apply when filtering
291	Found	r	Check success of one of the Find methods
292	IsUniDirectional	r	Is the dataset unidirectional (i.e. forward scrolling only)
292	Modified	r	Was the current record modified ?
302	OnCalcFields	rw	Event triggered when values for calculated fields must be computed.
302	OnDeleteError	rw	Event triggered when a delete operation fails.
302	OnEditError	rw	Event triggered when an edit operation fails.
303	OnFilterRecord	rw	Event triggered to filter records.
303	OnNewRecord	rw	Event triggered when a new record is created.
304	OnPostError	rw	Event triggered when a post operation fails.
293	RecNo	rw	Current record number
292	RecordCount	r	Number of records in the dataset
293	RecordSize	r	Size of the record in memory
293	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 (268) 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 (??)

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 (??), TDataSet.Create (??)

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 (??)

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 (??)

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 (??)` 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 (??), TDataSet.Edit (??)

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 (??)`) and attempts to post the record using `TDataSet.Post (??)`. 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 (??), TDataSet.SetFields (??), TDataSet.Post (??)

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` (??), `TDataSet.SetBookmark` (??), `TDataSet.FreeBookmark` (??), `TDataSet.BookmarkAvailable` (??)

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` (??) and `TDataSet.OnAfterCancel` (??) events. If the dataset was in insert mode, then the `TDataSet.OnBeforeScroll` (??) and `TDataSet.OnAfterScroll` (??) 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` (??), `TDataSet.Append` (??), `TDataSet.Insert` (??), `TDataSet.Edit` (??)

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` (235) exception is raised.

See also: `TDataSet.State` (??)

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` (235) exception will be raised.

See also: `TDataset.State` (??), `TField.Clear` (??)

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` (??) and `TDataset.OnAfterClose` (??) 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` (??), `TDataset.Open` (??)

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` (??) call can be used to disable sending of data events to the controls. The sending can be re-enabled with `TDataset.EnableControls` (??). 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` (??), `TDataset.EnableControls` (??)

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` (??), `TDataset.GetBookmark` (??), `TDataset.SetBookmark` (??)

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` (??) and `TBlobField.SaveToStream` (??) 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` (??), `TBlobField.SaveToStream` (??)

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` (??), `TDataset.SetFieldData` (??)

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 (??)`, `TDataset.BeforeScroll (??)`, `TDataset.AfterDelete (??)` and `TDataset.AfterScroll (??)` 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 (235)` exception will be raised.

See also: `TDataset.Cancel (??)`, `TDataset.BeforeDelete (??)`, `TDataset.BeforeScroll (??)`, `TDataset.AfterDelete (??)`, `TDataset.AfterScroll (??)`

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 (??)`. 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 (??)`.

See also: `TDataset.EnableControls (??)`, `TDataset.ControlsDisabled (??)`

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 (??)` and `TDataset.AfterEdit (??)` 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 (??)` instead.

Errors: If the dataset is read-only or not opened, then an `EDatabaseError (235)` exception will be raised.

See also: `TDataset.State (??)`, `TDataset.EOF (??)`, `TDataset.BOF (??)`, `TDataset.Append (??)`, `TDataset.BeforeEdit (??)`, `TDataset.AfterEdit (??)`

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` (??) 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` (??).

See also: `TDataSet.DisableControls` (??), `TDataSet.ControlsDisabled` (??)

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` (??): 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` (235) exception will be raised.

See also: `TFields.FieldByName` (??), `TDataSet.FindField` (??)

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` (??): 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.FieldName` (??), `TFields.FindField` (??)

10.20.27 TDataSet.FindFirst

Synopsis: Find the first active record (deprecated)

Declaration: `function FindFirst : Boolean`

Visibility: `public`

Description: `FindFirst` positions the cursor on the first record (taking into account filtering), and returns `True` if the cursor position was changed. This method must be implemented by descendents of `TDataSet`: The implementation in `TDataSet` always returns `False`, indicating that the position was not changed.

This method is deprecated, use `TDataSet.First (??)` instead.

See also: `TDataSet.First (??)`, `TDataSet.FindLast (??)`, `TDataSet.FindNext (??)`, `TDataSet.FindPrior (??)`

10.20.28 TDataSet.FindLast

Synopsis: Find the last active record (deprecated)

Declaration: `function FindLast : Boolean`

Visibility: `public`

Description: `FindLast` positions the cursor on the last record (taking into account filtering), and returns `True` if the cursor position was changed. This method must be implemented by descendents of `TDataSet`: The implementation in `TDataSet` always returns `False`, indicating that the position was not changed.

This method is deprecated, use `TDataSet.Last (??)` instead.

See also: `TDataSet.Last (??)`, `TDataSet.FindFirst (??)`, `TDataSet.FindNext (??)`, `TDataSet.FindPrior (??)`

10.20.29 TDataSet.FindNext

Synopsis: Find the next active record (deprecated)

Declaration: `function FindNext : Boolean`

Visibility: `public`

Description: `FindNext` positions the cursor on the next record (taking into account filtering), and returns `True` if the cursor position was changed. This method must be implemented by descendents of `TDataSet`: The implementation in `TDataSet` always returns `False`, indicating that the position was not changed.

This method is deprecated, use `TDataSet.Next (??)` instead.

See also: `TDataSet.Next (??)`, `TDataSet.FindFirst (??)`, `TDataSet.FindLast (??)`, `TDataSet.FindPrior (??)`

10.20.30 TDataSet.FindPrior

Synopsis: Find the previous active record (deprecated)

Declaration: `function FindPrior : Boolean`

Visibility: `public`

Description: `FindPrior` positions the cursor on the previous record (taking into account filtering), and returns `True` if the cursor position was changed. This method must be implemented by descendents of `TDataset`: The implementation in `TDataset` always returns `False`, indicating that the position was not changed.

This method is deprecated, use `TDataset.Prior (??)` instead.

See also: `TDataset.Prior (??)`, `TDataset.FindFirst (??)`, `TDataset.FindLast (??)`, `TDataset.FindPrior (??)`

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 (??)` and `TDataset.AfterScroll (??)` events. After the action is completed, the `TDataset.BOF (??)` property will be `True`.

Errors: If the dataset is unidirectional or is closed, an `EDatabaseError (235)` exception will be raised.

See also: `TDataset.Prior (??)`, `TDataset.Last (??)`, `TDataset.Next (??)`, `TDataset.BOF (??)`, `TDataset.BeforeScroll (??)`, `TDataset.AfterScroll (??)`

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 (??)`. It should not be used on bookmarks obtained with the `TDataset.Bookmark (??)` 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 (??)`, `TDataset.Bookmark (??)`

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 (??)` 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 (??)` call. The bookmark will be `Nil` if the dataset is empty or not active.

This call is deprecated. Use the `TDataset.Bookmark (??)` property instead to get a bookmark.

See also: `TDataset.SetBookmark (??)`, `TDataset.FreeBookmark (??)`, `TDataset.Bookmark (??)`

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` descendents to copy the data for the current record to `Buffer`. `Buffer` must point to a memory area, large enough to contain the data for the record. If the data is copied successfully to the buffer, the function returns `True`. The `TDataSet` implementation is empty, and returns `False`.

See also: `TDataSet.ActiveBuffer` (??)

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` (235) exception will be raised.

See also: `TDataSet.GetFieldNames` (??), `TDataSet.FieldByName` (??), `TDataSet.FindField` (??)

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` (??), `TDataSet.FieldByName` (??), `TDataSet.FindField` (??)

10.20.37 TDataSet.GotoBookmark

Synopsis: Jump to bookmark

Declaration: `procedure GotoBookmark(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` (??) function.

This function is deprecated, use the `TDataSet.Bookmark` (??) property instead.

Errors: if `ABookmark` does not contain a valid bookmark, then an exception may be raised.

See also: `TDataSet.Bookmark` (??), `TDataSet.GetBookmark` (??), `TDataSet.FreeBookmark` (??)

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 (??)`, `BeforeScroll (??)`, `OnNewRecord (??)`, `AfterInsert (??)` and `AfterScroll (??)` 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 (??)` must be called.

Errors: If the dataset is read-only, calling `Insert` will result in an `EDatabaseError (235)`.

See also: `BeforeInsert (??)`, `BeforeScroll (??)`, `OnNewRecord (??)`, `AfterInsert (??)`, `AfterScroll (??)`, `TDataSet.Post (??)`, `TDataSet.Append (??)`

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 (??)`, `TDataSet.SetFieldValues (??)`

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 (??)` and `TDataSet.BOF (??)` are both `True`, and the dataset is not in insert mode.

See also: `TDataSet.EOF (??)`, `TDataSet.BOF (??)`, `TDataSet.State (??)`

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" (??)` points to the current dataset instance, or indirectly.

See also: `TDataSource.Dataset (??)`

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 (??)` property to navigate in the records of the data. By default, this property is set to `True`, but `TDataSet` descendents may set this property to `False` (for instance, unidirectional datasets), in which case `RecNo` should not be used to navigate through the data.

See also: `TDataSet.RecNo (??)`

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 (??)`. After a call to `Last`, the `TDataSet.EOF (??)` property will be `True`.

Calling this method will trigger the `TDataSet.BeforeScroll (??)` and `TDataSet.AfterScroll (??)` events.

See also: `TDataSet.First (??)`, `TDataSet.Next (??)`, `TDataSet.EOF (??)`, `TDataSet.BeforeScroll (??)`, `TDataSet.AfterScroll (??)`

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` (228).

If a record is found that matches the criteria, then the `locate` operation positions the cursor on this record, and returns `True`. If no record is found to match the criteria, `False` is returned, and the position of the cursor is unchanged.

The implementation in `TDataSet` always returns `False`. It is up to `TDataSet` descendents to implement this method and return an appropriate value.

See also: `TDataSet.Find (??)`, `TDataSet.Lookup (??)`, `TLocateOption` (228)

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 (??)` 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 (235)` exception will be raised.

See also: `TDataSet.Locate (??)`

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 (??)` and `TDataSet.AfterScroll (??)` 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 (235)` exception on unidirectional datasets.

See also: `TDataSet.RecNo (??)`, `TDataSet.BeforeScroll (??)`, `TDataSet.AfterScroll (??)`

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 (??)` and `TDataSet.AfterScroll (??)` events. If the dataset is located on the last known record (`EOF (??)` 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 (235)` exception.

See also: `TDataSet.MoveBy (??)`, `TDataSet.Prior (??)`, `TDataSet.Last (??)`, `TDataSet.BeforeScroll (??)`, `TDataSet.AfterScroll (??)`, `TDataSet.EOF (??)`

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 descendents `InternalOpen` method is called to actually fetch data and initialize field-
defs and field instances.
3. `BOF (??)` is set to `True`
4. Internal buffers are allocated and filled with data
5. If the dataset is empty, `EOF (??)` is set to `true`
6. `State (??)` is set to `dsBrowse`
7. The `AfterOpen (??)` 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 (??)`, `TDataSet.State (??)`, `TDataSet.BOF (??)`, `TDataSet.EOF (??)`, `TDataSet.BeforeOpen (??)`, `TDataSet.AfterOpen (??)`

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 (??)`, `Append (??)` or `TDataSet.Edit (??)` 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 (??)`, `AfterPost (??)` events. After the call to `Past`, the `State (??)` 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 (219)`) will result in an `EdatabaseError (235)` exception. If an exception occurs during the save operation, the `OnPostError (??)` event is triggered to handle the error.

See also: `Insert (??)`, `Append (??)`, `Edit (??)`, `OnPostError (??)`, `BeforePost (??)`, `AfterPost (??)`, `State (??)`

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` (??) and `TDataSet.AfterScroll` (??) events. If the dataset is located on the first record, (`BOF` (??) 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` (235) exception.

See also: `TDataSet.MoveBy` (??), `TDataSet.Next` (??), `TDataSet.First` (??), `TDataSet.BeforeScroll` (??), `TDataSet.AfterScroll` (??), `TDataSet.BOF` (??)

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` (??) and `AfterRefresh` (??) events.

Errors: Refreshing may fail if the underlying dataset descendent does not support it.

See also: `TDataSet.Close` (??), `TDataSet.Open` (??), `BeforeRefresh` (??), `AfterRefresh` (??)

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` (??) should be used. The `Resync` parameter indicates how the buffers should be refreshed.

See also: `TDataSet.Refresh` (??)

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` (??) property, and works it's way down the array.

Errors: If the dataset is not in edit mode, then an `EDatabaseError` (235) exception will be raised. If there are more values than fields, an `EListError` exception will be raised.

See also: `TDataset.Fields` (??)

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` (??) property is set to `True`. The `toOEM` parameter is set to `True` if the transliteration must happen from the used codepage to the codepage used for storage, and if it is set to `False` then the transliteration must happen from the native codepage to the storage codepage. This call must be overridden by descendents of `TDataset` to provide the necessary transliteration: `TDataset` just copies the contents of the `Src` buffer to the `Dest` buffer. The result must be the number of bytes copied to the destination buffer.

Errors: No checks are performed on the buffers.

See also: `TStringField.Transliterate` (??)

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` (??)

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 descendents of `TDataset`.

See also: `OnUpdateRecord` (??)

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

Declaration: `Property BlockReadSize : Integer`

Visibility: `public`

Access: `Read,Write`

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 (??)`. As soon as `TDataSet.Next (??)` is called, `BOF` will no longer be true.

See also: `TDataSet.EOF (??)`, `TDataSet.Next (??)`, `TDataSet.First (??)`

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` (??) and `TDataset.EnableControls` (??) 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` (??), `TDataset.EnableControls` (??)

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` (??), `TDataset.Append` (??), `TDataset.Delete` (??), `Tdataset.Edit` (??)

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` (304)

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` (??) 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` (??), `TDataSet.Fields` (??), `TField` (316)

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` (??) method is called.
2. The `TDataSet.Last` (??) method is called (which is equivalent to moving to the last record and calling `TDataSet.Next` (??)).
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` (??), `TDataSet.Next` (??), `TDataSet.Last` (??), `TDataSet.IsEmpty` (??)

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` (??), i.e. the number of fields. For a dataset with persistent fields (when `DefaultFields` (??) 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` (346)

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` (??) 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` (??), `TDataset.DefaultFields` (??), `TDataset.Fields` (??)

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` (??), `TDataset.FindLast` (??), `TDataset.FindNext` (??) or `TDataset.FindPrior` (??) operations was succesful.

See also: `TDataset.FindFirst` (??), `TDataset.FindLast` (??), `TDataset.FindNext` (??), `TDataset.FindPrior` (??)

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 (??)` or `Tdataset.Insert (??)`. It becomes `True` if a value was written to one of the fields of the dataset.

See also: `Tdataset.Edit (??)`, `TDataSet.Insert (??)`, `TDataSet.Append (??)`, `TDataSet.Cancel (??)`, `TDataSet.Post (??)`

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 (??)` or `TDataSet.Next (??)`), random positioning of the cursor, editing or filtering will result in an `EDatabaseError (235)`. 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 (??)`, `TDataSet.Next (??)`

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` descendent may choose not to fetch all records from the database when the dataset is opened. If this is the case, then the `RecordCount` will only reflect the number of records that have actually been fetched at the current time, and therefor the value will change as more records are fetched from the database.

Only when `Last` has been called (and the dataset has been forced to fetch all records returned by the database), will the value of `RecordCount` be equal to the number of records returned by the query.

In general, datasets based on in-memory data or flat files, will return the correct number of records in `RecordCount`.

See also: `TDataset.RecNo` (??)

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` descendents, for `TDataset` the property always returns -1.

This property should not be used if exact positioning is required. it is inherently unreliable.

See also: `TDataset.RecordCount` (??)

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` (??), `TDataset.RecordCount` (??), `TDataset.RecNo` (??)

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` (??), `TDataset.Edit` (??), `TDataset.Insert` (??), `TDataset.Append` (??), `TDataset.Post` (??), `TDataset.Cancel` (??)

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` (346) 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` (344), `TField` (316)

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:

```
FieldByName(FieldName).AsVariant
```

It can be read as well as written.

See also: `TFields.FieldByName` (??)

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` (??), `TDataset.FilterOptions` (??)

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` (??) 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` (??), `TDataset.FilterOptions` (??)

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` (??), 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` (??), `TDataset.FilterOptions` (??)

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` (??) is then `dsInactive`). Setting the `Active` property to `True` is equivalent to calling `TDataset.Open` (??), setting it to `False` is equivalent to calling `TDataset.Close` (??)

See also: `TDataset.State` (??), `TDataset.Open` (??), `TDataset.Close` (??)

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` (??) event.

See also: `TField.FieldKind` (??), `TDataset.OnCalcFields` (??)

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` (??), `TDataset.State` (??)

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` (??) 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` (??), `TDataSet.State` (??), `TDataSet.AfterScroll` (??)

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` (??), `TDataSet.State` (??)

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 its internal memory structures. It is in `dsInactive` state when this event is triggered.

See also: `TDataSet.BeforeClose` (??), `TDataSet.State` (??)

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 (??)` or `TDataset.Insert (??)` 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 (??)`, `TDataset.Append (??)`, `TDataset.Insert (??)`

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 (??)` 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 (??)`, `TDataset.AfterScroll (??)`, `TDataset.Append (??)`, `TDataset.Insert (??)`

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 (??)` 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 (??)`, `TDataset.Edit (??)`, `TDataset.State (??)`

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 (??)`, `TDataset.Edit (??)`, `TDataset.State (??)`

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` (??) 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` (??), `TDataSet.AfterPost` (??), `TDataSet.State` (??)

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` (??) 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` (??) event is triggered instead.

See also: `TDataSet.BeforePost` (??), `TDataSet.Post` (??), `TDataSet.State` (??), `TDataSet.OnPostError` (??)

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` (??) 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` (??), `TDataSet.Cancel` (??), `TDataSet.State` (??)

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 (??) operation was successfully completed, and the dataset is again in dsBrowse state.

See also: TDataSet.BeforeCancel (??), TDataSet.Cancel (??), TDataSet.State (??)

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 (??) 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 (??) event. If the dataset was in insert mode when the Delete method was called, then the event will not be called, as TDataSet.Cancel (??) is called instead.

See also: TDataSet.AfterDelete (??), TDataSet.Delete (??), TDataSet.BeforeScroll (??), TDataSet.Cancel (??), TDataSet.State (??)

10.20.94 TDataSet.AfterDelete

Synopsis:

Declaration: Property AfterDelete : TDataSetNotifyEvent

Visibility: public

Access: Read,Write

Description: AfterDelete is triggered after the successful completion of the TDataSet.Delete (??) operation, when the dataset is again in dsBrowse state. The event is followed by a TDataSet.AfterScroll (??) event.

See also: TDataSet.BeforeDelete (??), TDataSet.Delete (??), TDataSet.AfterScroll (??), TDataSet.State (??)

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` (??), `TDataset.Prior` (??), `TDataset.First` (??), `TDataset.Last` (??), but also with two of the editing operations: `TDataset.Insert` (??) and `TDataset.Delete` (??). Raising an exception in this event handler aborts the operation in progress.

See also: `TDataset.AfterScroll` (??), `TDataset.Next` (??), `TDataset.Prior` (??), `TDataset.First` (??), `TDataset.Last` (??), `TDataset.Insert` (??), `TDataset.Delete` (??)

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` (??), `TDataset.Prior` (??), `TDataset.First` (??), `TDataset.Last` (??), but also with two of the editing operations: `TDataset.Insert` (??) and `TDataset.Delete` (??) 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` (??), `TDataset.Next` (??), `TDataset.Prior` (??), `TDataset.First` (??), `TDataset.Last` (??), `TDataset.Insert` (??), `TDataset.Delete` (??), `TDataset.Open` (??)

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` (??) 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` (??), `TDataset.AfterRefresh` (??), `TDataset.State` (??)

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` (??) 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` (??), `TDataset.State` (??), `TDataset.BeforeRefresh` (??)

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` (??) 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` (??), `TField.Kind` (??)

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` (??) method fails to delete the record in the underlying database. The event handler can be used to indicate what the response to the failed delete should be. To this end, it gets the exception object passed to it (parameter `E`), and it can examine this object to return an appropriate action in the `DataAction` parameter. The following responses are supported:

daFailThe operation should fail (an exception will be raised)

daAbortThe operation should be aborted (edits are undone, and an `EAbort` exception is raised)

daRetryRetry the operation.

For more information, see also the description of the `TDataSetErrorEvent` (222) event handler type.

See also: `TDataSetErrorEvent` (222), `TDataSet.Delete` (??), `TDataSet.OnEditError` (??), `TDataSet.OnPostError` (??)

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 (??)` method fails to put the dataset in edit mode because the underlying database engine reported an error. The event handler can be used to indicate what the response to the failed edit operation should be. To this end, it gets the exception object passed to it (parameter `E`), and it can examine this object to return an appropriate action in the `DataAction` parameter. The following responses are supported:

daFailThe operation should fail (an exception will be raised)

daAbortThe operation should be aborted (edits are undone, and an `EAbort` exception is raised)

daRetryRetry the operation.

For more information, see also the description of the `TDatasetErrorEvent (222)` event handler type.

See also: `TDatasetErrorEvent (222)`, `TDataset.Edit (??)`, `TDataset.OnDeleteError (??)`, `TDataset.OnPostError (??)`

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 (??)` property is set to `True`. The event handler should set the `Accept` parameter to `True` if the current record should be accepted, or to `False` if it should be rejected. No methods that change the state of the dataset may be used during this event, and calculated fields or lookup field values are not yet available.

See also: `TDataset.Filter (??)`, `TDataset.Filtered (??)`, `TDataset.state (??)`

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 (??)` or `TDataset.Insert (??)` methods when the buffer for the new record's data has been allocated. This event can be used to set default value for some of the fields in the dataset. If an exception is raised during this event handler, the operation is cancelled and the dataset is put again in browse mode (`TDataset.State (??)` is again `dsBrowse`).

See also: `TDataset.Append (??)`, `TDataset.Insert (??)`, `TDataset.State (??)`

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 (??) method fails to post the changes in the dataset buffer to the underlying database, because the database engine reported an error. The event handler can be used to indicate what the response to the failed post operation should be. To this end, it gets the exception object passed to it (parameter E), and it can examine this object to return an appropriate action in the DataAction parameter. The following responses are supported:

daFailThe operation should fail (an exception will be raised)

daAbortThe operation should be aborted (edits are undone, and an EAbort exception is raised)

daRetryRetry the operation.

For more information, see also the description of the TDataSetErrorEvent (222) event handler type.

See also: TDataSetErrorEvent (222), TDataSet.Post (??), TDataSet.OnDeleteError (??), TDataSet.OnEditError (??)

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 (268) instance. Any database aware component should never communicate with a dataset directly. Instead, it should communicate with a TDataSource (304) 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 (268), TDataSource (304)

10.21.2 Method overview

Page	Property	Description
305	Create	Create a new instance of TDataSource
305	Destroy	Remove a TDataSource instance from memory
305	Edit	Put the dataset in edit mode, if needed
306	IsLinkedTo	Check if a dataset is linked to a certain dataset

10.21.3 Property overview

Page	Property	Access	Description
306	AutoEdit	rw	Should the dataset be put in edit mode automatically
306	DataSet	rw	Dataset this datasource is connected to
307	Enabled	rw	Enable or disable sending of events
307	OnDataChange	rw	Called whenever data changes in the current record
307	OnStateChange	rw	Called whenever the state of the dataset changes
308	OnUpdateData	rw	Called whenever the data in the dataset must be updated
306	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` (??)

10.21.5 TDataSource.Destroy

Synopsis: Remove a `TDataSource` instance from memory

Declaration: `destructor Destroy; Override`

Visibility: `public`

Description: `Destroy` notifies all `TDataLink` ([263](#)) 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` (??), `TDataLink` ([263](#))

10.21.6 TDataSource.Edit

Synopsis: Put the dataset in edit mode, if needed

Declaration: `procedure Edit`

Visibility: `public`

Description: `Edit` will check `AutoEdit` (??): if it is `True`, then it puts the `Dataset` (??) 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` (??) directly.

Errors: An `EDatabaseError` ([235](#)) 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` (??), `TDataset.Edit` (??), `TDataset.State` (??)

10.21.7 TDataSource.IsLinkedTo

Synopsis: Check if a dataset is linked to a certain dataset

Declaration: `function IsLinkedTo (ADataset: TDataSet) : Boolean`

Visibility: public

Description: `IsLinkedTo` checks if it is somehow linked to `ADataset`: it checks the `Dataset` (??) 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` (??)

10.21.8 TDataSource.State

Synopsis: State of the dataset

Declaration: `Property State : TDataSetState`

Visibility: public

Access: Read

Description: `State` contains the `State` (??) 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` (??) directly, to take into account the effect of the `Enabled` (??) property.

See also: `TDataSet.State` (??), `TDataSource.Enabled` (??)

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` (??) 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` (??), `TDataSet.Edit` (??)

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 application programmer to the `TDataset` (268) 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` (268), `TDatasource.Enabled` (??)

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` (268), `TDatasource.Dataset` (??), `TDatasource.AutoEdit` (??)

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` (??) 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` (??), `TDatasource.OnStateChange` (??), `TDataset.State` (??), `TDatasource.Enabled` (??)

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` (??), `TDatasource.OnStateChange` (??), `TDataset.AfterScroll` (??), `TField.OnChange` (??), `TDatasource.Enabled` (??)

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` (??) 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` (??), `TDatasource.OnStateChange` (??), `TDataset.Post` (??)

10.22 TDateField

10.22.1 Description

`TDateField` is the class used when a dataset must manage data of type date. (`TField.DataType` (??) equals `ftDate`). It initializes some of the properties of the `TField` (316) 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` (268), `TField` (316), `TDateTimeField` (308), `TTimeField` (396)

10.22.2 Method overview

Page	Property	Description
308	Create	Create a new instance of a <code>TDateField</code> 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` (316) properties to configure the instance for working with date values.

See also: `TField` (316)

10.23 TDateTimeField

10.23.1 Description

`TDateTimeField` is the class used when a dataset must manage data of type datetime. (`TField.DataType` (??) equals `ftDateTime`). It also serves as base class for the `TDateField` (308) or `TTimeField` (396) classes. It overrides some of the properties and methods of the `TField` (316) 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` (268), `TField` (316), `TDateField` (308), `TTimeField` (396)

10.23.2 Method overview

Page	Property	Description
309	Create	Create a new instance of a <code>TDateTimeField</code> class.

10.23.3 Property overview

Page	Property	Access	Description
309	DisplayFormat	rw	Formatting string for textual representation of the field
310	EditMask		Specify an edit mask for an edit control
309	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` (316) properties to configure the instance for working with date/time values.

See also: `TField` (316)

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` (??) by `TDateTimeField` as a `TDateTime` value. It returns the same value as the `TField.AsDateTime` (??) property.

See also: `TField.AsDateTime` (??), `TField.Value` (??)

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` (??) property to format the value with the `DateTimeToString` (??) function.

See also: `DateTimeToString` (??), `FormatDateTime` (??), `TField.DisplayText` (??)

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 `EditMask` property, it is introduced in `TField`.

For more information on valid edit masks, see the documentation of the GUI controls.

See also: `TField.EditMask` (??)

10.24 TDBDataset

10.24.1 Description

`TDBDataset` is a `TDataset` descendent which introduces the concept of a database: a central component (`TDatabase` (258)) which represents a connection to a database. This central component is exposed in the `TDBDataset.Database` (??) 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` (??) 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` (258), `TDBTransaction` (311)

10.24.2 Method overview

Page	Property	Description
311	<code>destroy</code>	Remove the <code>TDBDataset</code> instance from memory.

10.24.3 Property overview

Page	Property	Access	Description
311	<code>DataBase</code>	rw	Database this dataset is connected to
311	<code>Transaction</code>	rw	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 (??) and Transaction (??). After this it calls the inherited destructor.

See also: TDBDataset.Database (??), TDatabase (258)

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 (258) 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 (235) exception will be raised.

See also: TDatabase (258), TDBDataset.Transaction (??)

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 (311) 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 (311), TDBDataset.Database (??)

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 (258) instance, and to keep track of TDataset instances which are connected to the transaction.

See also: TDatabase (258), TDataset (268)

10.25.2 Method overview

Page	Property	Description
312	<code>CloseDataSets</code>	Close all connected datasets
312	<code>Create</code>	Transaction property
312	<code>destroy</code>	Remove a <code>TDBTransaction</code> instance from memory.

10.25.3 Property overview

Page	Property	Access	Description
313	<code>Active</code>	rw	Is the transaction active or not
313	<code>DataBase</code>	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` (??)

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` ([310](#)) instances and then cleans up the resources allocated in the `Create` (??) constructor. After that it calls the inherited destructor.

See also: `TDBTransaction.Create` (??)

10.25.6 TDBTransaction.CloseDataSets

Synopsis: Close all connected datasets

Declaration: `procedure CloseDataSets`

Visibility: `public`

Description: `CloseDatasets` closes all connected datasets (All `TDBDataset` ([310](#)) instances whose `Transaction` (??) property points to this `TDBTransaction` instance).

See also: `TDBDataset` ([310](#)), `TDBDataset.Transaction` (??)

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 (235) exception.

See also: TDBTransaction.Active (??), TDataBase (258)

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 (??). Reversely, setting Active to True will call StartTransaction, setting it to False will call TDBTransaction.EndTransaction (??).

See also: TDBTransaction.StartTransaction (??), TDBTransaction.EndTransaction (??)

10.26 TDefCollection

10.26.1 Description

TDefCollection is a parent class for the TFieldDefs (344) and TIndexDefs (361) collections: It holds a set of named definitions on behalf of a TDataset (268) component. To this end, it introduces a dataset (??) property, and a mechanism to notify the dataset of any updates in the collection. It is supposed to hold items of class TNamedItem (373), so the TDefCollection.Find (??) method can find items by named.

10.26.2 Method overview

Page	Property	Description
314	create	Instantiate a new TDefCollection instance.
314	Find	Find an item by name
314	GetItemNames	Return a list of all names in the collection
314	IndexOf	Find location of item by name

10.26.3 Property overview

Page	Property	Access	Description
315	Dataset	r	Dataset this collection manages definitions for.
315	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` (268), `TNamedItem` (373)

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` (??), `TDefCollection.IndexOf` (??)

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` (??)

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` (??), `TNamedItem.Name` (??)

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 TDetailDataLink

10.27.1 Description

`TDetailDataLink` handles the communication between a detail dataset and the master datasource in a master-detail relationship between datasets. It should never be used in an application, and should only be used by component writers that wish to provide master-detail functionality for `TDataSet` descendents.

See also: `TDataSet` (268), `TDataSource` (304)

10.27.2 Property overview

Page	Property	Access	Description
315	<code>DetailDataSet</code>	r	Detail dataset in Master-detail relation

10.27.3 TDetailDataLink.DetailDataSet

Synopsis: Detail dataset in Master-detail relation

Declaration: `Property DetailDataSet : TDataSet`

Visibility: public

Access: Read

Description: `DetailDataSet` is the detail dataset in a master-detail relationship between 2 datasets. `DetailDataSet` is always `Nil` in `TDetailDataLink` and is only filled in in descendent classes like `TMasterDataLink` (369). The master dataset is available through the regular `TDataLink.DataSource` (??) property.

See also: `TDataSet` (268), `TMasterDataLink` (369), `TDataLink.DataSource` (??)

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` (268) 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` (268), `TFieldDef` (340), `TFields` (346)

10.28.2 Method overview

Page	Property	Description
319	<code>Assign</code>	Copy properties from one <code>TField</code> instance to another
319	<code>AssignValue</code>	Assign value of a variant record to the field.
320	<code>Clear</code>	Clear the field contents.
319	<code>Create</code>	Create a new <code>TField</code> instance
319	<code>Destroy</code>	Destroy the <code>TField</code> instance
320	<code>FocusControl</code>	Set focus to the first control connected to this field.
320	<code>GetData</code>	Get the data from this field
321	<code>IsBlob</code>	Is the field a BLOB field (untyped data of indeterminate size).
321	<code>IsValidChar</code>	Check whether a character is valid input for the field
321	<code>RefreshLookupList</code>	Refresh the lookup list
321	<code>SetData</code>	Save the field data
322	<code>SetFieldType</code>	Set the field data type
322	<code>Validate</code>	Validate the data buffer

10.28.3 Property overview

Page	Property	Access	Description
333	Alignment	rw	Alignment for this field
322	AsBCD	rw	Access the field's contents as a BCD (Binary coded Decimal)
323	AsBoolean	rw	Access the field's contents as a Boolean value.
323	AsBytes	rw	
323	AsCurrency	rw	Access the field's contents as a Currency value.
323	AsDateTime	rw	Access the field's contents as a TDateTime value.
324	AsFloat	rw	Access the field's contents as a floating-point (Double) value.
325	AsInteger	rw	Access the field's contents as a 32-bit signed integer (longint) value.
324	AsLargeInt	rw	Access the field's contents as a 64-bit signed integer (longint) value.
324	AsLongint	rw	Access the field's contents as a 32-bit signed integer (longint) value.
325	AsString	rw	Access the field's contents as an AnsiString value.
326	AsVariant	rw	Access the field's contents as a Variant value.
325	AsWideString	rw	Access the field's contents as a WideString value.
326	AttributeSet	rw	Not used: dictionary information
326	Calculated	rw	Is the field a calculated field ?
327	CanModify	r	Can the field's contents be modified.
333	ConstraintErrorMessage	rw	Message to display if the CustomConstraint constraint is violated.
327	CurValue	r	Current value of the field
333	CustomConstraint	rw	Custom constraint for the field's value
327	DataSet	rw	Dataset this field belongs to
327	DataSetSize	r	Size of the field's data
328	DataType	r	The data type of the field.
334	DefaultExpression	rw	Default value for the field
334	DisplayLabel	rws	Name of the field for display purposes
328	DisplayName	r	User-readable fieldname
328	DisplayText	r	Formatted field value
334	DisplayWidth	rw	Width of the field in characters
328	EditMask	rw	Specify an edit mask for an edit control
329	EditMaskPtr	r	Alias for EditMask
334	FieldKind	rw	The kind of field.
335	FieldName	rw	Name of the field
329	FieldNo	r	Number of the field in the record
335	HasConstraints	r	Does the field have any constraints defined
335	ImportedConstraint	rw	Constraint for the field value on the level of the underlying database
335	Index	rw	Index of the field in the list of fields
329	IsIndexedField	r	Is the field an indexed field ?
330	IsNull	r	Is the field empty
336	KeyFields	rw	Key fields to use when looking up a field value.
330	Lookup	rw	Is the field a lookup field
336	LookupCache	rw	Should lookup values be cached
336	LookupDataSet	rw	Dataset with lookup values
337	LookupKeyFields	rw	Names of fields on which to perform a locate
332	LookupList	r	List of lookup values
337	LookupResultField	rw	Name of field to use as lookup value
330	NewValue	rw	The new value of the field
330	Offset	r	Offset of the field's value in the dataset buffer
332	OldValue	r	Old value of the field
339	OnChange	rw	Event triggered when the field's value has changed
339	OnGetText	rw	Event to format the field's content
339	OnSetText	rw	Event to set the field's content based on a user-formatted string
339	OnValidate	rw	Event to validate the value of a field before it is writ-

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 (??) method will create the necessary TField instances, if none had been created in the designer.

See also: TDataSet.Open (??)

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 (??)

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 (??)

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` (235) exception.

See also: `TField.Assign` (??), `TField.Value` (??)

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` (??) 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` (235) exception.

See also: `TField.IsNull` (??), `TField.Value` (??)

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` (??), `TDataset.DisableControls` (??)

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` (??), `TField.SetData` (??), `TField.Value` (??)

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` (??)

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 specified by the `TField.ValidChars` (??) 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` (??)

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` (??) 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` (??), `LookupKeyFields` (??), `LookupResultField` (??)

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` (??), `TField.Value` (??)

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` (??)

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` (??) event handler, if one is set, to allow the application programmer to program additional checks.

See also: `TField.SetData` (??), `TField.OnValidate` (??)

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` (??), `TField.Value` (??)

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 (??)`, `TField.AsInteger (??)`

10.28.19 TField.AsBytes

Declaration: `Property AsBytes : TBytes`

Visibility: `public`

Access: `Read,Write`

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 (??)`, `TField.AsFloat (??)`

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). Therefore, when reading or writing a field value for a field whose native data type is not a `TDateTime`-compatible value (dates or string values), an exception may be raised.

See also: `TField.Value (??)`, `TField.AsString (??)`

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). Therefore, when reading or writing a field value for a field whose native data type is not a floating-point-compatible value (string values for instance), an exception may be raised.

See also: `TField.Value (??)`, `TField.AsString (??)`, `TField.AsCurrency (??)`

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). Therefore, when reading or writing a field value for a field whose native data type is not a 32-bit signed integer-compatible value (string values for instance), an exception may be raised.

This is an alias for the `TField.AsInteger (??)`.

See also: `TField.Value (??)`, `TField.AsString (??)`, `TField.AsInteger (??)`

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). Therefore, when reading or writing a field value for a field whose native data type is not a 64-bit signed integer-compatible value (string values for instance), an exception may be raised.

See also: `TField.Value` (??), `TField.AsString` (??), `TField.AsInteger` (??)

10.28.25 TField.AsInteger

Synopsis: Access the field's contents as a 32-bit signed integer (longint) value.

Declaration: `Property AsInteger : Integer`

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). Therefore, when reading or writing a field value for a field whose native data type is not a 32-bit signed integer-compatible value (string values for instance), an exception may be raised.

See also: `TField.Value` (??), `TField.AsString` (??), `TField.AsLongint` (??), `TField.AsInt64` (??)

10.28.26 TField.AsString

Synopsis: Access the field's contents as an `AnsiString` value.

Declaration: `Property AsString : string`

Visibility: public

Access: Read,Write

Description: `AsString` can be used to read or write the contents of the field as an `AnsiString` value. If the native type of the field is not an `ansistring` value, then an attempt will be made to convert the field value from the native format to 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). Therefore, when reading or writing a field value for a field whose native data type is not an `ansistring`-compatible value, an exception may be raised.

See also: `TField.Value` (??), `TField.AsWideString` (??)

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: `AsString` can be used to read or write the contents of the field as a `WideString` value. If the native type of the field is not a widestring value, then an attempt will be made to convert the field value from the native format to a widestring value when reading the field's content. Likewise, when writing the property, the value will be converted to the native type of the field (if the value allows it). Therefore, when reading or writing a field value for a field whose native data type is not a widestring-compatible value, an exception may be raised.

See also: `TField.Value` (??), `TField.AString` (??)

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). Therefore, when reading or writing a field value for a field whose native data type is not a `variant-compatible` value, an exception may be raised.

See also: `TField.Value` (??), `TField.AString` (??)

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` (??) 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` (??)

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 (??)`, `TDataset.CanModify (??)`

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 (??)`

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 (268)`, `TDataset.Fields (??)`

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 (??)` property which declares a logical size for datatypes that have a variable size (such as string fields). For BLOB fields, use the `TBlobField.BlobSize (??)` property to get the size of the field's contents for the current record..

See also: `TField.Size (??)`, `TBlobField.BlobSize (??)`

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` (??)

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` (??) property, unless assigned another value.

The use of this property is deprecated. Use `DisplayLabel` (??) instead.

See also: `TField.FieldName` (??)

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` (??) property, which does not take into account any formatting.

See also: `TField.AsString` (??)

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` (??), `TStringField.EditMask` (??)

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` (??) property. It is not used.

See also: `TField.EditMask` (??)

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` (??)

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` (268) can change this to `True`.

See also: `TField.Calculated` (??)

10.28.42 TField.IsNull

Synopsis: Is the field empty

Declaration: `Property IsNull : Boolean`

Visibility: `public`

Access: `Read`

Description: `IsNull` is `True` if the field does not have a value. If the underlying data contained a value, or a value is written to it, `IsNull` will return `False`. After `TDataset.Insert (??)` is called or `Clear (??)` is called then `IsNull` will return `True`.

See also: `TField.Clear (??)`, `TDataset.Insert (??)`

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 (??)` equals `fkLookup`, `False` otherwise. Setting the `Lookup` property will switch the `FieldKind` between the `fkLookup` and `fkData`.

See also: `TField.FieldKind (??)`

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` (268) does not yet support this.

See also: `TField.Value (??)`, `TField.CurValue (??)`

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 (??)`, `TField.Index (??)`, `TField.Datasize (??)`

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 (??)` should be used. For blob fields, the current size of the

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 (??)` event handler if a handler was specified. For display purposes, the `TField.DisplayText (??)` 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 (??)` property, this will invoke the `TField.OnSetText (??)` event handler, if one is set.

See also: `TField.AsString (??)`, `TField.DisplayText (??)`, `TField.Value (??)`

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 (??)` function.

See also: `TField.IsValidChar (??)`

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 (??)` 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 (??)`, `TField.Text (??)`, `TField.DisplayText (??)`

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 (??)`, `TField.CurValue (??)`, `TField.NewValue (??)`

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 (??)` property is set to `True`. It can be refreshed using the `RefreshLookupList (??)` method.

See also: `TField.RefreshLookupList (??)`, `TField.LookupCache (??)`

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` (??)

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 it's data. It should be a SQL-like expression that results in a `True` or `False` value. Examples of valid constraints are:

```
Salary < 10000
YearsEducation < Age
```

If the constraint is not satisfied when the record is posted, then an exception will be raised with the value of `ConstraintErrorMessage` (??) as a message.

This feature is not yet implemented in FPC.

See also: `TField.ConstraintErrorMessage` (??), `TField.ImportedConstraint` (??)

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` (??) is violated.

This feature is not yet implemented in FPC.

See also: `TField.CustomConstraint` (??)

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 (??)` or `TDataset.Insert (??)` 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 (??)`, `TDataset.Append (??)`, `TDataset.CustomConstraint (??)`

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 (??)` property, unless assigned another value.

See also: `TField.FieldName (??)`

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 (??)` for string fields) but can be modified to obtain a more appropriate value for the field's expected content.

See also: `TField.Alignment (??)`, `TField.DisplayText (??)`

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` (??), `TField.Size` (??), `TField.DataType` (??)

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` (??) or `ImportedConstraint` (??) properties is set to a non-empty value.

See also: `CustomConstraint` (??), `ImportedConstraint` (??)

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 data structures, but this need not always be so. The `TField.FieldNo` (??) property should be used for that.

See also: `TField.FieldNo` (??)

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` (268) descendent.

See also: `TField.CustomConstraint` (??), `TDataset` (268), `TField.ConstraintErrorMessage` (??)

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` (??) in the dataset pointed to by the `LookupDataset` (??) property.

See also: `LookupKeyFields` (??), `LookupDataset` (??)

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` (??) in the dataset pointed to by the `LookupDataset` (??) property. The list of key, value pairs is available through the `TField.LookupList` (??) property.

See also: `LookupKeyFields` (??), `LookupDataset` (??), `TField.LookupList` (??)

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` (??) property is used as a list of fields to locate a record in this dataset, and the value of the `LookupResultField` (??) field is then used as the value of the lookup field.

See also: `KeyFields` (??), `LookupKeyFields` (??), `LookupResultField` (??), `LookupCache` (??)

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` (??) property. These fields will be used when locating a record corresponding to the values in the `TField.KeyFields` (??) property.

See also: `KeyFields` (??), `LookupDataset` (??), `LookupResultField` (??), `LookupCache` (??)

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` (??) 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` (??), `LookupDataset` (??), `LookupKeyFields` (??), `LookupCache` (??)

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` (268) descendent or the database engine support retrieval of this property. It can be used to automatically create update statements, together with the `TField.ProviderFlags` (??) property.

See also: `TDataset` (268), `TField.ProviderFlags` (??)

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` (??)

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` (??)

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` (268) 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` (??), `ReadOnly` (??), `Visible` (??)

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` (??), `TField.Required` (??)

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 (??)` 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 (??)`

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 (??)` or `TField.DisplayText (??)` 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 (??)`, `TField.DisplayText (??)`, `TField.OnSetText (??)`, `TFieldGetTextEvent (224)`

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 (??)` 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 (??)` handler, if it exists.

See also: `TField.Text (??)`, `TField.OnGetText (??)`, `TFieldGetTextEvent (224)`

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 (??), TField.OnGetText (??), TField.OnSetText (??), TField.OnChange (??)

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 (??), TFieldDefs (344)

10.29.2 Method overview

Page	Property	Description
341	Assign	Assign the contents of one TFieldDef instance to another.
340	Create	Constructor for TFieldDef.
341	CreateField	Create TField instance based on definitions in current TFieldDef instance.
341	Destroy	Free the TFieldDef instance

10.29.3 Property overview

Page	Property	Access	Description
343	Attributes	rw	Additional attributes of the field.
343	DataType	rw	Data type for the field
342	FieldClass	r	TField class used for this fielddef
342	FieldNo	r	Field number
342	InternalCalcField	rw	Is this a definition of an internally calculated field ?
343	Precision	rw	Precision used in BCD (Binary Coded Decimal) fields
342	Required	rw	Is the field required ?
343	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` (??), `datatype` (??), `size` (??), `precision` (??), `FieldNo` (??) and the `Required` (??) property are all set according to the passed arguments.

Errors: If a duplicate name is passed, then an exception will occur.

See also: `Name` (??), `datatype` (??), `size` (??), `precision` (??), `FieldNo` (??), `Required` (??)

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` (??)

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` (340), 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` (??) what `TField` (316) descendent it should create, and then returns a newly created instance of this class. It sets the appropriate defaults for the `Size` (??), `FieldName` (??), `FieldNo` (??), `Precision` (??), `ReadOnly` (??) and `Required` (??) properties of the newly created instance. It should nver 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` (218) array is used to determine which `TField` Descendent class should be used when creating the `TField` instance, but descendents of `TDataset` may override the values in that array.

See also: `DefaultFieldClasses` (218), `TField` (316)

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 (??)` 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` (268), `CreateField (??)`, `TField` (316)

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 provided 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` (??)

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` (??), `TFieldDef.CreateField` (??)

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` (??)

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, BCD types), the `Size` property tells the buffer mechanism how many bytes are needed to hold the data for the field.

See also: `TFieldDef.Precision` (??), `TFieldDef.DataType` (??)

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 ([340](#)), TDataSet ([268](#))

10.30.2 Method overview

Page	Property	Description
344	Add	Add a new field definition to the collection.
345	AddFieldDef	Add new TFieldDef
345	Assign	Copy all items from one dataset to another
344	Create	Create a new instance of TFieldDefs
345	Find	Find item by name
346	MakeNameUnique	Create a unique field name starting from a base name
345	Update	Force update of definitions

10.30.3 Property overview

Page	Property	Access	Description
346	HiddenFields	rw	Should field instances be created for hidden fields
346	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 ([340](#)), TDataSet ([268](#))

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 (??)

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 (??)

10.30.7 TFieldDefs.Assign

Synopsis: Assign 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 (??)

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 (??) to find an item with name AName and typecasts the result to TFieldDef.

See also: TDefCollection.Find (??), TNamedItem.Name (??)

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 (??)

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` (??), `TFieldDef.Name` (??)

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` (??)

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` (340)

10.31 Tfields

10.31.1 Description

`TFields` mimics a `TCollection` class for the `Fields` (??) property of `TDataset` (268) instance. Since `TField` (316) is a descendent of `TComponent`, it cannot be an item of a collection, and must be managed by another class.

See also: `TField` (316), `TDataset` (268), `TDataset.Fields` (??)

10.31.2 Method overview

Page	Property	Description
347	Add	Add a new field to the list
348	CheckFieldName	Check field name for duplicate entries
348	CheckFieldNames	Check a list of field names for duplicate entries
348	Clear	Clear the list of fields
347	Create	Create a new instance of TFields
347	Destroy	Free the TFields instance
349	FieldByName	Find a field based on its name
349	FieldByNumber	Search field based on its fieldnumber
349	FindField	Find a field based on its name
349	GetEnumerator	Return an enumerator for the <code>for..in</code> construct
350	GetFieldNames	Get the list of fieldnames
350	IndexOf	Return the index of a field instance
350	Remove	Remove an instance from the list

10.31.3 Property overview

Page	Property	Access	Description
350	Count	r	Number of fields in the list
351	Dataset	r	Dataset the fields belong to
351	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 (??)` property, and initializes an internal list object to store the list of fields.

See also: `TDataSet (268)`, `TFields.Dataset (??)`, `TField (316)`

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 (??)`. After that it cleans up the internal structures and then calls the inherited destructor.

See also: `TDataSet (268)`, `TField (316)`, `TFields.Clear (??)`

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 (316) 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 (316), TFields.Clear (??)

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 (??)). If it does, then an EDatabaseError (235) exception is raised.

See also: TField.FieldName (??), TFields.Find (??)

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 (??).

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 (??), TFields.CheckFieldName (??), TFields.Find (??)

10.31.9 Tfields.Clear

Synopsis: Clear the list of fields

Declaration: `procedure Clear`

Visibility: public

Description: Clear removes all TField (316) var instances from the list. All field instances are freed after they have been removed from the list.

See also: TField (316)

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` (??) property matches `Value`. The search is performed case-insensitively. If no field instance is found, then `Nil` is returned.

See also: `TFields.FieldName` (??)

10.31.11 Tfields.FieldByName

Synopsis: Find a field based on its name

Declaration: `function FieldByName(const Value: string) : TField`

Visibility: public

Description: `Fieldbyname` searches the list of fields and returns the field instance whose `FieldName` (??) 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` (??), where `Nil` is returned if no match is found.

See also: `TFields.FindField` (??), `TFields.FieldName` (??), `TFields.FieldByNumber` (??), `TFields.IndexOf` (??)

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` (??) property matches the `FieldNo` parameter. If no such field is found, `Nil` is returned.

See also: `TFields.FieldByName` (??), `TFields.FindField` (??), `TFields.IndexOf` (??)

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` (351)

See also: `TFieldsEnumerator` (351), `#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` (??)

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` (??), `TFields.FieldByNumber` (??), `TFields.FindField` (??)

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` (??)

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` (??) property are numbered from 0 to `Count-1`.

See also: `TFields.fields` (??)

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` (346) 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` (??), `TField.Dataset` (??), `TDataSet` (268)

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` (??)

10.32 TFieldsEnumerator

10.32.1 Description

`TFieldsEnumerator` implements all the methods of `IEnumerator` so a `TFields` (346) 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` (316), `TFields` (346), `#rtl.system.IEnumerator` (??)

10.32.2 Method overview

Page	Property	Description
352	Create	Create a new instance of <code>TFieldsEnumerator</code> .
352	MoveNext	Move the current field to the next field in the collection.

10.32.3 Property overview

Page	Property	Access	Description
352	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` ([346](#)) 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` ([??](#))

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` ([??](#))

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 (??), MaxValue (??), MinValue (??) and overrides some TField (316) methods to work with floating point data.

It should never be necessary to create an instance of TFloatField manually, a field of this class will be instantiated automatically for each floating-point field when a dataset is opened.

See also: Currency (??), MaxValue (??), MinValue (??)

10.33.2 Method overview

Page	Property	Description
353	CheckRange	Check whether a value is in the allowed range of values for the field
353	Create	Create a new instance of the TFloatField

10.33.3 Property overview

Page	Property	Access	Description
354	Currency	rw	Is the field a currency field.
354	MaxValue	rw	Maximum value for the field
355	MinValue	rw	Minimum value for the field
355	Precision	rw	Precision (number of digits) of the field in text representations
354	Value	rw	Value of the field as a double type

10.33.4 TFloatField.Create

Synopsis: Create a new instance of the TFloatField

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

Visibility: public

Description: Create initializes a new instance of TFloatField. It calls the inherited constructor and then initializes some properties.

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 (??) and MaxValue (??) properties. If the value lies outside of the allowed range, then False is returned.

See also: MaxValue (??), MinValue (??)

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 (??)`

See also: `TField.AsFloat (??)`, `TField.Value (??)`

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 (??)` and `TField.Text (??)` 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 (??)`, `TField.Text (??)`, `TNumericField.DisplayFormat (??)`, `TNumericField.EditFormat (??)`

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 (??)` should also be set, because it will also be checked.

See also: `TFloatField.MinValue (??)`

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` (??) should also be set, because it will also be checked.

See also: `TFloatField.MaxValue` (??), `TFloatField.CheckRange` (??)

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` (??) or `TField.Text` (??), it is used in the arguments to `FormatFloat` (??).

See also: `TField.Displaytext` (??), `TField.Text` (??), `FormatFloat` (??)

10.34 TFMTBCDField

10.34.1 Method overview

Page	Property	Description
356	<code>CheckRange</code>	
355	<code>Create</code>	

10.34.2 Property overview

Page	Property	Access	Description
356	<code>Currency</code>	rw	
356	<code>MaxValue</code>	rw	
356	<code>MinValue</code>	rw	
356	<code>Precision</code>	rw	
356	<code>Size</code>		
356	<code>Value</code>	rw	

10.34.3 TFMTBCDField.Create

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

Visibility: public

10.34.4 TFMTBCDField.CheckRange

Declaration: `function CheckRange(AValue: TBCD) : Boolean`

Visibility: `public`

10.34.5 TFMTBCDField.Value

Declaration: `Property Value : TBCD`

Visibility: `public`

Access: `Read,Write`

10.34.6 TFMTBCDField.Precision

Declaration: `Property Precision : LongInt`

Visibility: `published`

Access: `Read,Write`

10.34.7 TFMTBCDField.Currency

Declaration: `Property Currency : Boolean`

Visibility: `published`

Access: `Read,Write`

10.34.8 TFMTBCDField.MaxValue

Declaration: `Property MaxValue : string`

Visibility: `published`

Access: `Read,Write`

10.34.9 TFMTBCDField.MinValue

Declaration: `Property MinValue : string`

Visibility: `published`

Access: `Read,Write`

10.34.10 TFMTBCDField.Size

Declaration: `Property Size :`

Visibility: `published`

Access:

10.35 TGraphicField

10.35.1 Description

`TGraphicField` is the class used when a dataset must manage graphical BLOB data. (`TField.DataType` (??) equals `ftGraphic`). It initializes some of the properties of the `TField` (316) class. All methods to be able to work with graphical BLOB data have been implemented in the `TBlobField` (244) parent class.

It should never be necessary to create an instance of `TGraphicField` manually, a field of this class will be instantiated automatically for each graphical BLOB field when a dataset is opened.

See also: `TDataset` (268), `TField` (316), `TBLOBField` (244), `TMemoField` (372), `TWideMemoField` (398)

10.35.2 Method overview

Page	Property	Description
357	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` (316) properties to configure the instance for working with graphical BLOB values.

See also: `TField` (316)

10.36 TGUIDField

10.36.1 Description

`TGUIDField` is the class used when a dataset must manage native variant-typed data. (`TField.DataType` (??) equals `ftGUID`). It initializes some of the properties of the `TField` (316) 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` (268), `TField` (316), `TGUIDField.AsGuid` (??)

10.36.2 Method overview

Page	Property	Description
358	Create	Create a new instance of the <code>TGUIDField</code> class

10.36.3 Property overview

Page	Property	Access	Description
358	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` ([316](#)) properties to configure the instance for working with GUID values.

See also: `TField` ([316](#))

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: `TField.AsString` (??)

10.37 TIndexDef

10.37.1 Description

`TIndexDef` describes one index in a set of indexes of a `TDataset` ([268](#)) instance. The collection of indexes is described by the `TIndexDefs` ([361](#)) class. It just has the necessary properties to describe an index, but does not implement any functionality to maintain an index.

See also: `TIndexDefs` ([361](#))

10.37.2 Method overview

Page	Property	Description
359	Create	Create a new index definition

10.37.3 Property overview

Page	Property	Access	Description
360	CaseInsFields	rw	Fields in field list that are ordered case-insensitively
360	DescFields	rw	Fields in field list that are ordered descending
359	Expression	rw	Expression that makes up the index values
359	Fields	rw	Fields making up the index
360	Options	rw	Index options
360	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` (358) 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` (361) 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` (361), `TIndexDef.Options` (??), `TIndexDef.Fields` (??)

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` (??)

See also: `TIndexDef.Options` (??), `TIndexDef.Fields` (??)

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` (??) property can be used to determine the fields in the list that have a descending sort order. The `TIndexDef.CaseInsFields` (??) property determines which fields are sorted in a case-insensitive manner.

See also: `TIndexDef.DescFields` (??), `TIndexDef.CaseInsFields` (??), `TIndexDef.Expression` (??)

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 (??)` property which are ordered in a case-insensitive manner. `CaseInsFields` may not contain fieldnames that do not appear in `Fields`.

See also: `TIndexDef.Fields (??)`, `TIndexDef.Expression (??)`, `TIndexDef.DescFields (??)`

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 (??)` property which are ordered in a descending manner. `DescFields` may not contain fieldnames that do not appear in `Fields`.

See also: `TIndexDef.Fields (??)`, `TIndexDef.Expression (??)`, `TIndexDef.DescFields (??)`

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` ([228](#)) for more information on the various available options.

See also: `TIndexOptions` ([228](#))

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` (358), `TIndexDefs.Items` (??)

10.38.2 Method overview

Page	Property	Description
361	<code>Add</code>	Add a new index definition with given name and options
362	<code>AddIndexDef</code>	Add a new, empty, index definition
361	<code>Create</code>	Create a new <code>TIndexDefs</code> instance
362	<code>Find</code>	Find an index by name
362	<code>FindIndexForFields</code>	Find index definition based on field names
362	<code>GetIndexForFields</code>	Get index definition based on field names
363	<code>Update</code>	Called whenever one of the items changes

10.38.3 Property overview

Page	Property	Access	Description
363	<code>Items</code>	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` (358).

See also: `TIndexDef` (358), `TIndexDefs.Destroy` (??)

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` (358) 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` (358), `TNamedItem.Name` (??), `TIndexDef.Fields` (??), `TIndexDef.Options` (??), `TIndexDefs.AddIndexDef` (??)

10.38.6 TIndexDefs.AddIndexDef

Synopsis: Add a new, empty, index definition

Declaration: `function AddIndexDef : TIndexDef`

Visibility: public

Description: `AddIndexDef` adds a new `TIndexDef` (358) 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` (??)

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` (??) method to search and return a `TIndexDef` (358) instance based on the name. The search is case-insensitive and returns `Nil` if no matching index definition was found.

See also: `TIndexDef` (358), `TDefCollection.Find` (??), `TIndexDefs.FindIndexForFields` (??)

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` (??) 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` (358), `TIndexDefs.Find` (??), `TIndexDefs.GetIndexForFields` (??)

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` (??) 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` (358), `TIndexDefs.Find` (??), `TIndexDefs.FindIndexForFields` (??)

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` ([358](#))

10.39 TIntegerField

10.39.1 Description

`TIntegerField` is an alias for `TLongintField` ([365](#)).

See also: `TLongintField` ([365](#)), `TField` ([316](#))

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` ([316](#)) 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` (??) and `MaxValue` (??).

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` ([316](#)), `MinValue` (??), `MaxValue` (??)

10.40.2 Method overview

Page	Property	Description
364	<code>CheckRange</code>	Check whether a values falls within the allowed range
364	<code>Create</code>	Create a new instance of the <code>TLargeintField</code> class

10.40.3 Property overview

Page	Property	Access	Description
364	MaxValue	rw	Maximum value for the field
365	MinValue	rw	Minimum value for the field
364	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` ([316](#)) and `MinValue` (??) and `MaxValue` (??).

See also: `TField` ([316](#)), `MinValue` (??), `MaxValue` (??)

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` (??) and `MaxValue` (??) properties. If the value lies outside of the allowed range, then `False` is returned.

See also: `MaxValue` (??), `MinValue` (??)

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` (??).

See also: `TField.Value` (??), `TField.AsLargeInt` (??)

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` (??) should also be set, because it will also be checked.

See also: `TLargeIntField.MinValue` (??)

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` (??) should also be set, because it will also be checked.

See also: `TLargeIntField.MaxValue` (??)

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` (316) 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` (??) and `MaxValue` (??).

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` (316), `MaxValue` (??), `MinValue` (??)

10.41.2 Method overview

Page	Property	Description
366	<code>CheckRange</code>	Check whether a valid is in the allowed range of values for the field
366	<code>Create</code>	Create a new instance of <code>TLongintField</code>

10.41.3 Property overview

Page	Property	Access	Description
366	<code>MaxValue</code>	rw	Maximum value for the field
367	<code>MinValue</code>	rw	Minimum value for the field
366	<code>Value</code>	rw	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` (??) and `MaxValue` (??) properties.

See also: TField (316), `MaxValue` (??), `MinValue` (??)

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` (??) and `MaxValue` (??) properties. If the value lies outside of the allowed range, then `False` is returned.

See also: `MaxValue` (??), `MinValue` (??)

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` (??) property.

See also: TField.Value (??)

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` (??)

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` (??)

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` (??)

10.42.2 Method overview

Page	Property	Description
368	<code>Add</code>	Add a key, value pair to the list
368	<code>Clear</code>	Remove all key, value pairs from the list
367	<code>Create</code>	Create a new instance of <code>TLookupList</code> .
367	<code>Destroy</code>	Free a <code>TLookupList</code> instance from memory
368	<code>FirstKeyByValue</code>	Find the first key that matches a value
368	<code>ValueOfKey</code>	Look up value based on a key
369	<code>ValuesToStrings</code>	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` (??)

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` (??)

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` (??)

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` (??)

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` (??)

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` (??), `TLookupList.Add` (??)

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 drop-down list.

See also: `TLookupList.ValueOfKey (??)`

10.43 TMasterDataLink

10.43.1 Description

`TMasterDataLink` is a `TDataLink` descendent which handles master-detail relations. It can be used in `TDataset` (268) descendents that must have master-detail functionality: the detail dataset creates an instance of `TMasterDataLink` to point to the master dataset, which is subsequently available through the `TDataLink.Dataset (??)` property.

The class also provides functionailty for keeping a list of fields that make up the master-detail functionality, in the `TMasterDataLink.FieldNameNames (??)` and `TMasterDataLink.Fields (??)` properties.

This class should never be used in application code.

See also: `TDataset` (268), `TDataLink.DataSource (??)`, `TDataLink.DataSet (??)`, `TMasterDataLink.FieldNameNames (??)`, `TMasterDataLink.Fields (??)`

10.43.2 Method overview

Page	Property	Description
369	Create	Create a new instance of <code>TMasterDataLink</code>
370	Destroy	Free the datalink instance from memory

10.43.3 Property overview

Page	Property	Access	Description
370	FieldNames	rw	List of fieldnames that make up the master-detail relationship
370	Fields	r	List of fields as specified in <code>FieldNames</code>
370	OnMasterChange	rw	Called whenever the master dataset data changes
371	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 (??)` property. The master dataset must be set through the `DataSource (??)` property, and is usually set by the applictaion programmer.

See also: `TDetailDataLink.DetailDataset` (??), `TDatalink.Datasource` (??)

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` (??)

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` (??)) on which the master-detail relationship is based. Setting this property will fill the `TMasterDataLink.Fields` (??) property with the field instances of the master dataset.

See also: `TMasterDataLink.Fields` (??), `TDatalink.Dataset` (??), `TDataset.GetFieldList` (??)

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` (316) instances from the master dataset (`TDatalink.Dataset` (??)) when the `FieldNames` (??) property is set, and when the master dataset opens.

See also: `TField` (316), `TMasterDatalink.FieldNames` (??)

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` (??) list is empty, `TMasterDataLink.OnMasterDisable` (??) is called instead.

See also: `TMasterDataLink.OnMasterDisable` (??)

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 (??)

10.44 TMasterParamsDataLink

10.44.1 Description

TMasterParamsDataLink is a TDataLink (263) 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 (263), TDataSet (268), TParams (388), TParam (376)

10.44.2 Method overview

Page	Property	Description
372	CopyParamsFromMaster	Copy parameter values from master dataset.
371	Create	Initialize a new TMasterParamsDataLink instance
372	RefreshParamNames	Refresh the list of parameter names

10.44.3 Property overview

Page	Property	Access	Description
372	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 (388) in the published properties of ADataset and assigns it to the Params (??) property.

See also: TDataSet (268), TParams (388), TMasterParamsDataLink.Params (??)

10.44.5 TMasterParamsDataLink.RefreshParamNames

Synopsis: Refresh the list of parameter names

Declaration: `procedure RefreshParamNames; Virtual`

Visibility: `public`

Description: `RefreshParamNames` scans the `Params` (??) property and sets the `FieldNames` (??) property to the list of parameter names.

See also: `TMasterParamsDataLink.Params` (??), `TMasterDataLink.FieldNames` (??)

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` (??), 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` (??) 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` (??), `TParam.Bound` (??)

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` (388), `TMasterParamsDataLink.Create` (??)

10.45 TMemoField

10.45.1 Description

`TMemoField` is the class used when a dataset must manage memo (Text BLOB) data. (`TField.DataType` (??) equals `ftMemo`). It initializes some of the properties of the `TField` (316) class. All methods to be able to work with memo fields have been implemented in the `TBlobField` (244) 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` (268), `TField` (316), `TBLOBField` (244), `TWideMemoField` (398), `TGraphicField` (357)

10.45.2 Method overview

Page	Property	Description
373	Create	Create a new instance of the <code>TMemoField</code> class

10.45.3 Property overview

Page	Property	Access	Description
373	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` ([316](#)) properties to configure the instance for working with memo values.

See also: `TField` ([316](#))

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` (??) with a default value of `true`.

See also: `TBlobField.Transliterate` (??), `TStringField.Transliterate` (??), `TDataset.Translate` (??)

10.46 `TNamedItem`

10.46.1 Description

`NamedItem` is a `TCollectionItem` (??) descendent which introduces a `Name` (??) property. It automatically returns the value of the `Name` property as the value of the `DisplayName` (??) property.

See also: `DisplayName` (??), `Name` (??)

10.46.2 Property overview

Page	Property	Access	Description
374	<code>DisplayName</code>	<code>rw</code>	Display name
374	<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` (??) property.

See also: `Name` (??)

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` (??) property. If the `TNamedItem` item is owned by a `TDefCollection` (313) collection, then the name must be unique, i.e. each `Name` value may appear only once in the collection.

See also: `DisplayName` (??), `TDefCollection` (313)

10.47 TNumericField

10.47.1 Description

`TNumericField` is an abstract class which overrides some of the methods of `TField` (316) 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` (??) and `TNumericField.EditFormat` (??).

Since `TNumericField` is an abstract class, it must never be instantiated directly. Instead one of the descendent classes should be created.

See also: `TField` (316), `TNumericField.DisplayFormat` (??), `TNumericField.EditFormat` (??), `TField.Alignment` (??), `TIntegerField` (363), `TLargeIntField` (363), `TFloatField` (353), `TBCDField` (240)

10.47.2 Method overview

Page	Property	Description
375	Create	Create a new instance of <code>TNumericField</code>

10.47.3 Property overview

Page	Property	Access	Description
375	Alignment		Alignment of the field
375	DisplayFormat	rw	Format string for display of numerical data
375	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 (??) property with

See also: TField.Alignment (??)

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 (??)

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 (??) 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 (??), TNumericField.EditFormat (??)

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 (??) 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 (??), TNumericField.DisplayFormat (??)

10.48 TParam

10.48.1 Description

`TParam` is one item in a `TParams` (388) collection. It describes the name (`TParam.Name` (??)), type (`ParamType` (??)) and value (`TParam.Value` (??)) 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` (268) descendent that owns the parameters should have created all necessary `TParam` instances.

See also: `TParams` (388)

10.48.2 Method overview

Page	Property	Description
377	<code>Assign</code>	Assign one parameter instance to another
378	<code>AssignField</code>	Copy value from field instance
378	<code>AssignFieldValue</code>	Assign field value to the parameter.
379	<code>AssignFromField</code>	Copy field type and value
378	<code>AssignToField</code>	Assign parameter value to field
379	<code>Clear</code>	Clear the parameter value
377	<code>Create</code>	Create a new parameter value
379	<code>GetData</code>	Get the parameter value from a memory buffer
379	<code>GetDataSize</code>	Return the size of the data.
380	<code>LoadFromFile</code>	Load a parameter value from file
380	<code>LoadFromStream</code>	Load a parameter value from stream
380	<code>SetBlobData</code>	Set BLOB data
380	<code>SetData</code>	Set the parameter value from a buffer

10.48.3 Property overview

Page	Property	Access	Description
381	AsBlob	rw	Return parameter value as a blob
381	AsBoolean	rw	Get/Set parameter value as a boolean value
381	AsCurrency	rw	Get/Set parameter value as a currency value
382	AsDate	rw	Get/Set parameter value as a date (TDateTime) value
382	AsDateTime	rw	Get/Set parameter value as a date/time (TDateTime) value
382	AsFloat	rw	Get/Set parameter value as a floating-point value
384	AsFMTBCD	rw	
382	AsInteger	rw	Get/Set parameter value as an integer (32-bit) value
383	AsLargeInt	rw	Get/Set parameter value as a 64-bit integer value
383	AsMemo	rw	Get/Set parameter value as a memo (string) value
383	AsSmallInt	rw	Get/Set parameter value as a smallint value
383	AsString	rw	Get/Set parameter value as a string value
384	AsTime	rw	Get/Set parameter value as a time (TDateTime) value
386	AsWideString	rw	Get/Set the value as a widestring
384	AsWord	rw	Get/Set parameter value as a word value
384	Bound	rw	Is the parameter value bound (set to fixed value)
385	Dataset	r	Dataset to which this parameter belongs
386	DataType	rw	Data type of the parameter
385	IsNull	r	Is the parameter empty
386	Name	rw	Name of the parameter
385	NativeStr	rw	No description available
387	NumericScale	rw	Numeric scale
387	ParamType	rw	Type of parameter
387	Precision	rw	Precision of the BCD value
388	Size	rw	Size of the parameter
385	Text	rw	Read or write the value of the parameter as a string
386	Value	rws	Value as a variant

10.48.4 TParam.Create

Synopsis: Create a new parameter value

Declaration: `constructor Create(ACollection: TCollection); Override; Overload
 constructor Create(AParams: TParams; AParamType: TParamType); Overload
 ; Reintroduce`

Visibility: public

Description: `Create` first calls the inherited `create`, and then initializes the parameter properties. The first form creates a default parameter, the second form is a convenience function and initializes a parameter of a certain kind (`AParamType`), in which case the owning `TParams` collection must be specified in `AParams`

See also: `TParams` ([388](#))

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 (316), then it is passed to TParam.AssignField (??). If Source is of type TStrings, then it is assigned to TParams.AsMemo (??).

Errors: If Source is not of type TParam, TField or TStrings, an exception will be raised.

See also: TField (316), TParam.Name (??), TParam.Bound (??), TParam.NumericScale (??), TParam.ParamType (??), TParam.value (??), TParam.Size (??), TParam.AssignField (??), Tparam.AsMemo (??)

10.48.6 TParam.AssignField

Synopsis: Copy value from field instance

Declaration: procedure AssignField(Field: TField)

Visibility: public

Description: AssignField copies the Field, FieldName (??) and Value (??) 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 (??), TParam.AssignToField (??), TParam.AssignFieldValue (??)

10.48.7 TParam.AssignToField

Synopsis: Assign parameter value to field

Declaration: procedure AssignToField(Field: TField)

Visibility: public

Description: AssignToField copies the parameter value (??) to the field instance. If Field is Nil, nothing happens.

Errors: An EDatabaseError (235) exception is raised if the field has an unsupported field type (for types ftCursor, ftArray, ftDataset, ftReference).

See also: TParam.Assign (??), TParam.AssignField (??), TParam.AssignFromField (??)

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 (??) bound parameter to True. This method is called from TParam.AssignField (??).

See also: TField (316), TParam.AssignField (??), TParam.Bound (??)

10.48.9 TParam.AssignFromField

Synopsis: Copy field type and value

Declaration: `procedure AssignFromField(Field: TField)`

Visibility: public

Description: `AssignFromField` copies the field value (??) and data type (`TField.DataType` (??)) to the parameter instance. If `Field` is `Nil`, nothing happens. This is the reverse operation of `TParam.AssignToField` (??).

Errors: An `EDatabaseError` (235) exception is raised if the field has an unsupported field type (for types `ftCursor`, `ftArray`, `ftDataset`, `ftReference`).

See also: `TParam.Assign` (??), `TParam.AssignField` (??), `TParam.AssignToField` (??)

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` (??), `TParam.Name` (??), `TParam.ParamType` (??), `TParam.DataType` (??)

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` (316), 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` (235) exception.

See also: `TParam.SetData` (??), `TField` (316)

10.48.12 TParam.GetDataSize

Synopsis: Return the size of the data.

Declaration: `function GetDataSize : Integer`

Visibility: public

Description: `GetDataSize` returns the size (in bytes) needed to store the current value of the parameter.

Errors: For an unsupported data type, an `EDatabaseError` (235) exception is raised when this function is called.

See also: `TParam.GetData` (??), `TParam.SetData` (??)

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` (??).

Errors: If the specified `FileName` is not a valid file, or the file is not readable, an exception will occur.

See also: `TParam.LoadFromStream` (??), `TBlobType` (220), `TParam.SaveToFile` (??)

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` (??), `TParam.SaveToStream` (??)

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` (??)

10.48.16 TParam.SetData

Synopsis: Set the parameter value from a buffer

Declaration: `procedure SetData(Buffer: Pointer)`

Visibility: public

Description: `SetData` performs the reverse operation of `TParam.GetData (??)`: it reads the parameter value from the memory area pointed to by `Buffer`. The size of the data read is determined by `TParam.GetDataSize (??)` and the type of data by `TParam.DataType (??)`: it is the same storage mechanism used by `TField (316)`, and so can be used to copy the value from a `TDataset (268)` record buffer.

Errors: Not all field types are supported. If an unsupported field type is encountered, an `EDatabaseError (235)` exception is raised.

See also: `TDataset (268)`, `TParam.GetData (??)`, `TParam.DataType (??)`, `TParam.GetDataSize (??)`

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: `TParam.AsString (??)`

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 (??)`, `TParam.Value (??)`

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 (??)`, `TParam.DataType (??)`, `TParam.Value (??)`

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` (??), `TParam.AsTime` (??), `TParam.DataType` (??), `TParam.Value` (??)

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` (??), `TParam.asTime` (??), `TParam.DataType` (??), `TParam.Value` (??)

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` (??), `TParam.DataType` (??), `TParam.Value` (??)

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` (??), `TParam.AsSmallInt` (??), `TParam.AsWord` (??), `TParam.DataType` (??), `TParam.Value` (??)

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 (??)`, `TParam.asSmallint (??)`, `TParam.AsWord (??)`, `TParam.DataType (??)`, `TParam.Value (??)`

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 (??)`, `TParam.LoadFromStream (??)`, `TParam.SaveToStream (??)`, `TParam.DataType (??)`, `TParam.Value (??)`

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 (??)`, `TParam.AsLargeInt (??)`, `TParam.AsWord (??)`, `TParam.DataType (??)`, `TParam.Value (??)`

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 (??)`, `TParam.Value (??)`

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 (??)`, `TParam.AsDateTime (??)`, `TParam.DataType (??)`, `TParam.Value (??)`

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 (??)`, `TParam.AsLargeInt (??)`, `TParam.AsSmallint (??)`, `TParam.DataType (??)`, `TParam.Value (??)`

10.48.30 TParam.AsFmtBCD

Declaration: `Property AsFmtBCD : TBCD`

Visibility: `public`

Access: `Read,Write`

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 (??)`, `TParam.Value (??)`

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` (388) 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` (268), `TParams` (388)

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` (??), `TParam.Value` (??)

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` (??), but, when written, does not set the data type: instead, it attempts to convert the value to the type specified in `TParam.DataType` (??).

See also: `TParam.AsString` (??), `TParam.DataType` (??)

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` (??)

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` (??) to `ftWideString`.

See also: `TParam.AsString` (??), `TParam.Value` (??), `TParam.DataType` (??)

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` (??), `TParam.Value` (??), `TParam.AssignField` (??)

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` (??), `TParam.Value` (??), `TParams.ParamByName` (??)

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 (??)`, `TParam.Size (??)`

10.48.41 TParam.ParamType

Synopsis: Type of parameter

Declaration: `Property ParamType : TParamType`

Visibility: published

Access: Read,Write

Description: `ParamType` specifies the type of parameter: is the parameter value written to the database engine, or is it received from the database engine, or both ? It can have the following value:

ptUnknownUnknown type

ptInputInput parameter

ptOutputOutput paramete, filled on result

ptInputOutputInput/output parameter

ptResultResult parameter

The `ParamType` property is usually set by the database engine that creates the parameter instances.

See also: `TParam.DataType (??)`, `TParam.DataSize (??)`, `TParam.Name (??)`

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 (??)`, `TParam.Size (??)`

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` (??) in the `TParam.AssignFieldValue` (??) method. The actual size can be retrieved through the `TParam.Datasize` (??) property.

See also: `TParam.Datasize` (??), `TField.DataSize` (??), `TParam.AssignFieldValue` (??)

10.49 TParams

10.49.1 Description

`TParams` is a collection of `TParam` (376) 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` (376) 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` (??).

`TDataset` (268) 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` (371) 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` (??) method exists.

See also: `TDataset` (268), `TMasterParamsDataLink` (371), `TParam` (376), `TParams.ParseSQL` (??), `TParams.CopyParamValuesFromDataset` (??)

10.49.2 Method overview

Page	Property	Description
389	<code>AddParam</code>	Add a parameter to the collection
389	<code>AssignValues</code>	Copy values from another collection
392	<code>CopyParamValuesFromDataset</code>	Copy parameter values from a the fields in a dataset.
389	<code>Create</code>	Create a new instance of <code>TParams</code>
389	<code>CreateParam</code>	Create and add a new parameter to the collection
390	<code>FindParam</code>	Find a parameter with given name
390	<code>GetParamList</code>	Fetch a list of <code>TParam</code> instances
390	<code>IsEqual</code>	Is the list of parameters equal
391	<code>ParamByName</code>	Return a parameter by name
391	<code>ParseSQL</code>	Parse SQL statement, replacing parameter names with SQL parameter placeholders
392	<code>RemoveParam</code>	Remove a parameter from the collection

10.49.3 Property overview

Page	Property	Access	Description
392	Dataset	r	Dataset that owns the TParams instance
393	Items	rw	Indexed access to TParams instances in the collection
393	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` ([376](#)) 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` ([376](#))

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` ([376](#)), `#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` ([376](#)) 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` ([??](#))). If it is not found, nothing is done.

See also: `TParam` ([376](#)), `TParam.Assign` ([??](#))

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` (376) 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` (376), `TParam.Name` (??), `TParam.Datatype` (??), `TParam.Paramtype` (??)

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` (376) 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` (??), `TParams.ParamByName` (??), `TParams.GetParamList` (??)

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` (376), `TParam.Name` (??), `TParams.ParamByName` (??)

10.49.10 TParams.IsEqual

Synopsis: Is the list of parameters equal

Declaration: `function IsEqual(Value: TParams) : Boolean`

Visibility: public

Description: `IsEqual` compares the parameter count of `Value` and if it matches, it compares all `TParam` items of `Value` with the items it owns. If all items are equal (all properties match), then `True` is returned. The items are compared on index, so the order is important.

See also: `TParam` (376)

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` (376) 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` (??), `TParams.FindParam` (??), `TParams.GetParamList` (??)

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;`
`var ParamBinding: TParamBinding) : string; Overload`
`function ParseSQL(SQL: string; DoCreate: Boolean; EscapeSlash: Boolean;`
`EscapeRepeat: Boolean; ParameterStyle: TParamStyle;`
`var ParamBinding: TParamBinding;`
`var 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:

psInterbaseParameters are specified by a `?` character

psPostgreSQLParameters are specified by a `$N` character.

psSimulatedParameters are specified by a `$N` character.

`psInterbase` is the default.

If the `EscapeSlash` parameter is `True`, then backslash characters are used to quote the next character in the SQL statement. If it is `False`, the backslash character is regarded as a normal character.

If the `EscapeRepeat` parameter is `True` (the default) then embedded quotes in string literals are escaped by repeating themselves. If it is `false` then they should be quoted with backslashes.

`ParamBinding`, if specified, is filled with the indexes of the parameter instances in the parameter collection: for each SQL parameter placeholder, the index of the corresponding `TParam` instance is returned in the array.

`ReplaceString`, if specified, contains the placeholder used for the parameter names (by default, \$). It has effect only when `ParameterStyle` equals `psSimulated`.

The function returns the SQL statement with the parameter names replaced by placeholders.

See also: `TParam` (376), `TParam.Name` (??), `TParamStyle` (229)

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` (376)

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` (??). By default, this operation is only performed on `TParam` instances with their `Bound` (??) 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` (376), `TParam.Bound` (??), `TParam.AssignField` (??), `TDataSet` (268), `TDataSet.FieldName` (??)

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` (268) instance that was specified when the `TParams` instance was created.

See also: `TParams.Create` (??), `TDataSet` (268)

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 ([388](#))

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 (??)) by name. It is equivalent to reading and writing

`ParamByName (ParamName) .Value`

See also: TParam.Value (??), TParams.ParamByName (??)

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 ([316](#)), TNumericField ([374](#)), TLongintField ([365](#)), TWordField ([400](#))

10.50.2 Method overview

Page	Property	Description
394	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 (393) class. It calls the inherited constructor and then simply sets some of the TField (316) properties to work with 16-bit signed integer data.

See also: TField (316)

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 (316) 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 (316), TWideStringField (399), TDataset (268)

10.51.2 Method overview

Page	Property	Description
394	Create	Create a new instance of the TStringField class
395	SetFieldType	Set the field type

10.51.3 Property overview

Page	Property	Access	Description
396	EditMask		Specify an edit mask for an edit control
395	FixedChar	rw	Is the string declared with a fixed length ?
396	Size		Maximum size of the string
395	Transliterate	rw	Should the field value be transliterated when reading or writing
395	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 (316) 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` (394) to check the data type more accurately (`ftString` and `ftFixedChar`). No extra functionality is added.

See also: `TField.DataType` (??)

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` (??).

See also: `TDataset.Translate` (??)

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` (??) when read, or sets the `AsString` property when written to.

See also: `TField.AsString` (??), `TField.Value` (??)

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 `EditMask` property, it is introduced in `TField`.

For more information on valid edit masks, see the documentation of the GUI controls.

See also: `TField.EditMask` (??)

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` (??)

10.52 TTimeField

10.52.1 Description

`TimeField` is the class used when a dataset must manage data of type time. (`TField.DataType` (??) equals `ftTime`). It initializes some of the properties of the `TField` (316) 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` (268), `TField` (316), `TDateTimeField` (308), `TDateField` (308)

10.52.2 Method overview

Page	Property	Description
397	Create	Create a new instance of a <code>TTimeField</code> 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` (316) properties to configure the instance for working with time values.

See also: `TField` (316)

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` (??) equals `ftVarBytes`). It initializes some of the properties of the `TField` (316) class to be able to work with variable-size byte fields.

It should never be necessary to create an instance of `TVarBytesField` manually, a field of this class will be instantiated automatically for each variable-sized binary data field when a dataset is opened.

See also: `TDataset` (268), `TField` (316), `TBytesField` (250)

10.53.2 Method overview

Page	Property	Description
397	Create	Create a new instance of a <code>TVarBytesField</code> 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` (316) properties to configure the instance for working with variable-size binary data values.

See also: `TField` (316)

10.54 TVariantField

10.54.1 Description

`TVariantField` is the class used when a dataset must manage native variant-typed data. (`TField.DataType` (??) equals `ftVariant`). It initializes some of the properties of the `TField` (316) 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 (268), TField (316)

10.54.2 Method overview

Page	Property	Description
398	Create	Create a new instance of the TVariantField 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 (316) properties to configure the instance for working with variant values.

See also: TField (316)

10.55 TWideMemoField

10.55.1 Description

TWideMemoField is the class used when a dataset must manage memo (Text BLOB) data. (TField.DataType (??) equals ftWideMemo). It initializes some of the properties of the TField (316) class. All methods to be able to work with widestring memo fields have been implemented in the TBlobField (244) 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 (268), TField (316), TBlobField (244), TMemoField (372), TGraphicField (357)

10.55.2 Method overview

Page	Property	Description
398	Create	Create a new instance of the TWideMemoField class

10.55.3 Property overview

Page	Property	Access	Description
399	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` (316) properties to configure the instance for working with widestring memo values.

See also: `TField` (316)

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` (??) property.

See also: `TField.Value` (??), `Tfield.AsWideString` (??)

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` (316) 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` (316), `TStringField` (394), `TDataset` (268)

10.56.2 Method overview

Page	Property	Description
399	<code>Create</code>	Create a new instance of the <code>TWideStringField</code> class.
400	<code>SetFieldType</code>	Set the field type

10.56.3 Property overview

Page	Property	Access	Description
400	<code>Value</code>	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` (316) 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` (399) to check the data type more accurately (`ftWideString` and `ftFixedWideChar`). No extra functionality is added.

See also: `TField.DataType` (??)

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` (??) property.

See also: `TField.AsWideString` (??), `TField.Value` (??)

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` (316), `TNumericField` (374), `TLongintField` (365), `TSmallIntField` (393)

10.57.2 Method overview

Page	Property	Description
400	<code>Create</code>	Create a new instance of the <code>TWordField</code> 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` (400) class. It calls the inherited constructor and then simply sets some of the `TField` (316) properties to work with 16-bit unsigned integer data.

See also: `TField` (316)

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 (`dbugsvr`) which can be started in advance, and which writes debug message to the console (both on Windows and Linux). The Lazarus project contains a visual application which displays the messages in a GUI.

The `dbugintf` unit relies on the SimpleIPC (547) mechanism to communicate with the debug server, hence it works on all platforms that have a functional version of that unit. It also uses `TProcess` to start the debug server if needed, so the process (516) unit should also be functional.

11.2 Writing a debug server

Writing a debug server is relatively easy. It should instantiate a `TSimpleIPCServer` class from the SimpleIPC (547) 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 (547) mechanism to communicate with the debug server, hence it works on all platforms that have a functional version of that unit. It also uses `TProcess` to start the debug server if needed, so the process (516) unit should also be functional.

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 encounters 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
<code>dlError</code>	Error message
<code>dlInformation</code>	Informational message
<code>dlWarning</code>	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` (406), `SendDebug` (403)

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` (403) call will initialize the debug client when it is first called.

Errors: None.

See also: `SendDebug` (403), `StartDebugServer` (406)

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` (403) which sends the name and value of a boolean value as an informational message.

Errors: None.

See also: `SendDebug` (403), `SendDateTime` (403), `SendInteger` (405), `SendPointer` (406)

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` (403) 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` (403), `SendBoolean` (403), `SendInteger` (405), `SendPointer` (406)

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` (402).

See also: `SendDebugEx` (404), `SendDebugFmt` (404), `SendDebugFmtEx` (404)

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` (403) uses informational messages.

Other than that the function of `SendDebugEx` is equal to that of `SendDebug`

Errors: None.

See also: `SendDebug` (403), `SendDebugFmt` (404), `SendDebugFmtEx` (404)

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` (403). It exists mainly to avoid the `Format` call in calling code.

Errors: None.

See also: `SendDebug` (403), `SendDebugEx` (404), `SendDebugFmtEx` (404), `#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` (404) with `Debug level MType`. It exists mainly to avoid the `Format` call in calling code.

Errors: None.

See also: `SendDebug` (403), `SendDebugEx` (404), `SendDebugFmt` (404), `#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` (403) 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` (403), `SendBoolean` (403), `SendDateTime` (403), `SendPointer` (406)

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` (405), 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` (403), `SendMethodExit` (405), `SendSeparator` (406)

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` (405), 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` (403), `SendMethodEnter` (405), `SendSeparator` (406)

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` (403) 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` (403), `SendBoolean` (403), `SendDateTime` (403), `SendInteger` (405)

11.4.13 SendSeparator

Synopsis: Send a separator message

Declaration: `procedure SendSeparator`

Visibility: default

Description: `SendSeparator` is a simple wrapper around `SendDebug` (403) 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` (403), `SendMethodEnter` (405), `SendMethodExit` (405)

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` (403) 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` (402), `SendDebug` (403)

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` (403) call will attempt to start the server by itself if it is not yet running.

Errors: On error, `False` is returned.

See also: `SendDebug` (403), `InitDebugClient` (403)

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` (401) 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` (410) and `ReadDebugMessageFromStream` (409) 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` (409) 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` (409) 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` (409), `WriteDebugMessageToStream` (410)

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` (409) 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` (409), `ReadDebugMessageFromStream` (409)

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 ([413](#)) 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

SErrLogFailedMsg = '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

SLogError = '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 (??) 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 (??) and OnGetCustomEventID (??) 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 (??) 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 (413) component to indicate errors.

See also: TEventLog (413)

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 (??) call, or the specialized Warning (??), Error (??), Info (??) or Debug (??) calls, which have the event type predefined.

See also: Log (??), Warning (??), Error (??), Info (??), Debug (??)

13.5.2 Method overview

Page	Property	Description
416	Debug	Log a debug message
413	Destroy	Clean up TEventLog instance
416	Error	Log an error message to
414	EventTypeToString	Create a string representation of an event type
416	Info	Log an informational message
415	Log	Log a message to the system log.
415	Pause	
414	RegisterMessageFile	Register message file
415	Resume	
415	UnRegisterMessageFile	Unregister the message file (needed on windows only)
416	Warning	Log a warning message.

13.5.3 Property overview

Page	Property	Access	Description
417	Active	rw	Activate the log mechanism
417	AppendContent	rw	Control whether output is appended to an existing file
419	CustomLogType	rw	Custom log type ID
418	DefaultEventType	rw	Default event type for the Log (415) call.
419	EventIDOffset	rw	Offset for event ID messages identifiers
418	FileName	rw	File name for log file
417	Identification	rw	Identification string for messages
417	LogType	rw	Log type
420	OnGetCustomCategory	rw	Event to retrieve custom message category
420	OnGetCustomEvent	rw	Event to retrieve custom event Code
420	OnGetCustomEventID	rw	Event to retrieve custom event ID
420	Paused	rw	
418	RaiseExceptionOnError	rw	Determines whether logging errors are reported or ignored
419	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` (??) property to `False`.

See also: `Active` (??)

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

1 Should contain the description of the `etInfo` event type.

2 Should contain the description of the `etWarning` event type.

4 Should contain the description of the `etError` event type.

4 Should contain the description of the `etDebug` event type.

None of these strings should have a string substitution placeholder.

The second type of strings are the **message definitions**. Their number starts at `EventIDOffset` (??) (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 (??)`. 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 (??)`

13.5.8 TEventLog.Pause

Declaration: `procedure Pause`

Visibility: public

13.5.9 TEventLog.Resume

Declaration: `procedure Resume`

Visibility: public

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 `Default-EventType (??)`.

If `EventType` is `etCustom`, then the `OnGetCustomEvent (??)`, `OnGetCustomEventID (??)` and `OnGetCustomCategory (??)`.

The other logging calls: `Info (??)`, `Warning (??)`, `Error (??)` and `Debug (??)` use the `Log` call to do the actual work.

See also: `Info (??)`, `Warning (??)`, `Error (??)`, `Debug (??)`, `OnGetCustomEvent (??)`, `OnGetCustomEventID (??)`, `OnGetCustomCategory (??)`

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 (??)`, `Info (??)`, `Error (??)`, `Debug (??)`

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 (??)`, `Info (??)`, `Warning (??)`, `Debug (??)`

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 (??)`, `Info (??)`, `Warning (??)`, `Error (??)`

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 (??)`, `Warning (??)`, `Error (??)`, `Debug (??)`

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` (??), `FileName` (??)

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` (??), `TimeStampFormat` (??)

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` (??) property.

See also: `FileName` (??)

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 (??)`

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 (415)` call.

Declaration: `Property DefaultEventType : TEventType`

Visibility: published

Access: Read,Write

Description: `DefaultEventType` is the event type used by the `Log (??)` call if it's `EventType` parameter is omitted.

See also: `Log (??)`

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 (??)` 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 (??)`

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 (??)`

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 (??)` to format the custom log event type string.

See also: `EventTypeToString (??)`

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 (??)` 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 (??)`

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 (??), OnGetCustomEvent (??)

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 (??), OnGetCustomEvent (??)

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 (??), OnGetCustomEventID (??)

13.5.28 TEventLog.Paused

Declaration: Property Paused : Boolean

Visibility: published

Access: Read,Write

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` (??) or `DoGet` (??) 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` ([422](#))

See also: `TEZcgi` ([422](#))

14.5 TEZcgi

14.5.1 Description

`TEZcgi` implements all functionality to analyze the CGI environment and query the variables present in it. It's main use is the exposed variables.

Programs wishing to use this class should make a descendent class of this class and override the `DoPost` (??) or `DoGet` (??) methods. To run the program, an instance of this class must be created, and it's `Run` (??) 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
422	<code>Create</code>	Creates a new instance of the <code>TEZCGI</code> component
422	<code>Destroy</code>	Removes the <code>TEZCGI</code> component from memory
424	<code>DoGet</code>	Method to handle <code>GET</code> requests
424	<code>DoPost</code>	Method to handle <code>POST</code> requests
424	<code>GetValue</code>	Return the value of a request variable.
423	<code>PutLine</code>	Send a line of output to the web-client
423	<code>Run</code>	Run the CGI application.
423	<code>WriteContent</code>	Writes the content type to standard output

14.5.3 Property overview

Page	Property	Access	Description
426	<code>Email</code>	rw	Email of the server administrator
426	<code>Name</code>	rw	Name of the server administrator
425	<code>Names</code>	r	Indexed array with available variable names.
424	<code>Values</code>	r	Variables passed to the CGI script
426	<code>VariableCount</code>	r	Number of available variables.
425	<code>Variables</code>	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` (??) property.

See also: `Variables` (??), `Names` (??), `Values` (??)

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 (??)`

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 (??)` property) and calls the `DoPost (??)` or `DoGet (??)` 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 (??)`, `DoPost (??)`, `DoGet (??)`

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 (??)` or `DoGet (??)` methods.

See also: `DoPost (??)`, `DoGet (??)`, `PutLine (??)`

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 (??)` 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 (??)`

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 (??)`

14.5.10 TEZcgi.DoPost

Synopsis: Method to handle POST requests

Declaration: `procedure DoPost; Virtual`

Visibility: public

Description: `DoPost` is called by the `Run (??)` method the POST method was used to invoke the CGI application. It should be overridden in descendents of `TEZcgi` to actually handle the request.

See also: `Run (??)`, `DoGet (??)`

14.5.11 TEZcgi.DoGet

Synopsis: Method to handle GET requests

Declaration: `procedure DoGet; Virtual`

Visibility: public

Description: `DoGet` is called by the `Run (??)` method the GET method was used to invoke the CGI application. It should be overridden in descendents of `TEZcgi` to actually handle the request.

See also: `Run (??)`, `DoPost (??)`

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_TYPEAuthorization type

CONTENT_LENGTHContent length

CONTENT_TYPEContent type

GATEWAY_INTERFACEUsed gateway interface
PATH_INFORequested URL
PATH_TRANSLATEDTransformed URL
QUERY_STRINGClient query string
REMOTE_ADDRAddress of remote client
REMOTE_HOSTDNS name of remote client
REMOTE_IDENTRemote identity.
REMOTE_USERRemote user
REQUEST_METHODRequest methods (POST or GET)
SCRIPT_NAMEScript name
SERVER_NAMEServer host name
SERVER_PORTServer port
SERVER_PROTOCOLServer protocol
SERVER_SOFTWAREWeb server software
HTTP_ACCEPTAccepted responses
HTTP_ACCEPT_CHARSETAccepted character sets
HTTP_ACCEPT_ENCODINGAccepted encodings
HTTP_IF_MODIFIED_SINCEProxy information
HTTP_REFERERReferring page
HTTP_USER_AGENTClient 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` (??), `TEZCGI.Names` (??), `TEZCGI.GetValue` (??), `TEZcgi.VariableCount` (??)

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` (??). Any other value will result in an exception being raised.

See also: `TEZcgi.Variables` (??), `TEZcgi.Values` (??), `TEZcgi.GetValue` (??), `TEZcgi.VariableCount` (??)

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 (??)`. Any other value will result in an exception being raised.

See also: `TEZcgi.Names (??)`, `TEZcgi.Values (??)`, `TEZcgi.GetValue (??)`, `TEZcgi.VariableCount (??)`

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 (??)` and `Variables (??)` properties.

See also: `Names (??)`, `Variables (??)`, `TEZcgi.Values (??)`, `TEZcgi.GetValue (??)`

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 (??)` method.

See also: `TEZcgi.Run (??)`, `TEZcgi.Email (??)`

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 (??)` method.

See also: `TEZcgi.Run (??)`, `TEZcgi.Name (??)`

Chapter 15

Reference for unit 'fpTimer'

15.1 Used units

Table 15.1: Used units by unit 'fpTimer'

Name	Page
Classes	??
System	??

15.2 Overview

The `fpTimer` unit implements a timer class `TFPTimer` (429) which can be used on all supported platforms. The timer class uses a driver class `TFPTimerDriver` (430) 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` (427) variable to the class pointer of the driver class. The driver class should descend from `TFPTimerDriver` (430).

15.3 Constants, types and variables

15.3.1 Types

```
TFPTimerDriverClass = Class of TFPTimerDriver
```

`TFPTimerDriverClass` is the class pointer of `TFPTimerDriver` (430) it exists mainly for the purpose of being able to set `DefaultTimerDriverClass` (427), so a custom timer driver can be used for the timer instances.

15.3.2 Variables

```
DefaultTimerDriverClass : TFPTimerDriverClass = Nil
```

`DefaultTimerDriverClass` contains the `TFPTimerDriver` (430) class pointer that should be used when a new instance of `TFPCustomTimer` (428) 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.

15.4 TFPCustomTimer

15.4.1 Description

`TFPCustomTimer` is the timer class containing the timer's implementation. It relies on an extra driver instance (of type `TFPTimerDriver` (430)) to do the actual work.

`TFPCustomTimer` publishes no events or properties, so it is unsuitable for handling in an IDE. The `TFPTimer` (429) descendent class publishes all needed events of `TFPCustomTimer`.

See also: `TFPTimerDriver` (430), `TFPTimer` (429)

15.4.2 Method overview

Page	Property	Description
428	Create	Create a new timer
428	Destroy	Release a timer instance from memory
429	StartTimer	Start the timer
429	StopTimer	Stop the timer

15.4.3 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` (??)

15.4.4 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` (??)

15.4.5 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 (??)` event is called.

See also: `StopTimer (??)`, `Enabled (??)`, `OnTimer (??)`

15.4.6 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 (??)`, `Enabled (??)`, `OnTimer (??)`

15.5 TFPTimer

15.5.1 Description

`TFPTimer` implements no new events or properties, but merely publishes events and properties already implemented in `TFPCustomTimer` (428): `Enabled (??)`, `OnTimer (??)` and `Interval (??)`.

The `TFPTimer` class is suitable for use in an IDE.

See also: `TFPCustomTimer` (428), `Enabled (??)`, `OnTimer (??)`, `Interval (??)`

15.5.2 Property overview

Page	Property	Access	Description
429	<code>Enabled</code>		Start or stop the timer
430	<code>Interval</code>		Timer tick interval in milliseconds.
430	<code>OnTimer</code>		Event called on each timer tick.

15.5.3 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 (??)`), setting it to `False` will stop the timer (calling `StopTimer (??)`).

See also: `StartTimer (??)`, `StopTimer (??)`, `OnTimer (??)`, `Interval (??)`

15.5.4 TFPTimer.Interval

Synopsis: Timer tick interval in milliseconds.

Declaration: `Property Interval :`

Visibility: published

Access:

Description: `Interval` specifies the timer interval in milliseconds. Every `Interval` milliseconds, the `OnTimer (??)` 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 (??)`, `Enabled (??)`

15.5.5 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 (??)`, `Enabled (??)`

15.6 TFPTimerDriver

15.6.1 Description

`TFPTimerDriver` is the abstract timer driver class: it simply provides an interface for the `TFP-CustomTimer (428)` class to use.

The `fpTimer` unit implements a descendent of this class which implements the default timer mechanism.

See also: `TFPCustomTimer (428)`, `DefaultTimerDriverClass (427)`

15.6.2 Method overview

Page	Property	Description
431	Create	Create new driver instance
431	StartTimer	Start the timer
431	StopTimer	Stop the timer

15.6.3 Property overview

Page	Property	Access	Description
431	Timer	r	Timer tick

15.6.4 TFPTimerDriver.Create

Synopsis: Create new driver instance

Declaration: `constructor Create(ATimer: TFPCustomTimer); Virtual`

Visibility: `public`

Description: `Create` should be overridden by descendents of `TFPTimerDriver` to do additional initialization of the timer driver. `Create` just stores (in `Timer` (??)) a reference to the `ATimer` instance which created the driver instance.

See also: `Timer` (??), `TFPTimer` (429)

15.6.5 TFPTimerDriver.StartTimer

Synopsis: Start the timer

Declaration: `procedure StartTimer; Virtual; Abstract`

Visibility: `public`

Description: `StartTimer` is called by `TFPCustomTimer.StartTimer` (??). It should be overridden by descendents of `TFPTimerDriver` to actually start the timer.

See also: `TFPCustomTimer.StartTimer` (??), `TFPTimerDriver.StopTimer` (??)

15.6.6 TFPTimerDriver.StopTimer

Synopsis: Stop the timer

Declaration: `procedure StopTimer; Virtual; Abstract`

Visibility: `public`

Description: `StopTimer` is called by `TFPCustomTimer.StopTimer` (??). It should be overridden by descendents of `TFPTimerDriver` to actually stop the timer.

See also: `TFPCustomTimer.StopTimer` (??), `TFPTimerDriver.StartTimer` (??)

15.6.7 TFPTimerDriver.Timer

Synopsis: Timer tick

Declaration: `Property Timer : TFPCustomTimer`

Visibility: `public`

Access: `Read`

Description: `Timer` calls the `TFPCustomTimer` (428) timer event. Descendents of `TFPTimerDriver` should call `Timer` whenever a timer tick occurs.

See also: `TFPTimer.OnTimer` (??), `TFPTimerDriver.StartTimer` (??), `TFPTimerDriver.StopTimer` (??)

Chapter 16

Reference for unit 'gettext'

16.1 Used units

Table 16.1: Used units by unit 'gettext'

Name	Page
Classes	??
System	??
sysutils	??

16.2 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` ([434](#))) 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.

16.3 Constants, types and variables

16.3.1 Constants

```
MOFileHeaderMagic = $950412de
```

This constant is found as the first integer in a `.mo`

16.3.2 Types

```
PLongWordArray = ^TLongWordArray
```

Pointer to a `TLongWordArray` ([433](#)) array.

```
PMOStringTable = ^TMOStringTable
```

Pointer to a TMOStringTable (433) array.

```
PPCharArray = ^TPCharArray
```

Pointer to a TPCharArray (433) array.

```
TLongWordArray = Array[0..(1 shl 30) div SizeOf(LongWord)] of LongWord
```

TLongWordArray is an array used to define the PLongWordArray (432) 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.

```
TMOFileHeader = packed record
  magic : LongWord;
  revision : LongWord;
  nstrings : LongWord;
  OrigTabOffset : LongWord;
  TransTabOffset : LongWord;
  HashTabSize : LongWord;
  HashTabOffset : LongWord;
end
```

This structure describes the structure of a .mo file with string localizations.

```
TMOStringInfo = packed record
  length : LongWord;
  offset : LongWord;
end
```

This record is one element in the string tables describing the original and translated strings. It describes the position and length of the string. The location of these tables is stored in the TMOFileHeader (433) record at the start of the file.

```
TMOStringTable = Array[0..(1 shl 30) div SizeOf(TMOStringInfo)] of TMOStringInfo
```

TMOStringTable is an array type containing TMOStringInfo (433) records. It should never be used directly, as it would occupy too much memory.

```
TPCharArray = Array[0..(1 shl 30) div SizeOf(PChar)] of PChar
```

TLongWordArray is an array used to define the PPCharArray (433) 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.

16.4 Procedures and functions

16.4.1 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.

16.4.2 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` (434) 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` (434), `TMOFile` (434)

16.4.3 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` (434), 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` (434), `TMOFile` (434)

16.5 EMOFileError

16.5.1 Description

`EMOFileError` is raised in case an `TMOFile` (434) instance is created with an invalid .mo.

See also: `TMOFile` (434)

16.6 TMOFile

16.6.1 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.

16.6.2 Method overview

Page	Property	Description
435	Create	Create a new instance of the <code>TMOFile</code> class.
435	Destroy	Removes the <code>TMOFile</code> instance from memory
435	Translate	Translate a string

16.6.3 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` ([433](#)) structure, then an `EMOFileError` ([434](#)) exception is raised.

See also: `TMOFile.Destroy` (??)

16.6.4 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` (??)

16.6.5 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 17

Reference for unit 'IBConnection'

17.1 Used units

Table 17.1: Used units by unit 'IBConnection'

Name	Page
bufdataset	??
Classes	??
db	218
dbconst	??
ibase60dyn	??
sqldb	??
System	??
sysutils	??

17.2 Constants, types and variables

17.2.1 Constants

`DEFDIALECT` = 3

Default dialect that will be used when connecting to databases. See Dialect (??) for more details on dialects.

`MAXBLOBSEGMENTSIZE` = 65535

17.3 EIBDatabaseError

17.3.1 Description

Firebird/Interbase database error, a descendant of EDatabaseError ([235](#)).

See also: TSQLConnection.EDatabaseError (??)

17.4 TIBConnection

17.4.1 Description

`TIBConnection` is a descendant of `TSQLConnection` (??) and represents a connection to a Firebird/Interbase server.

It is designed to work with Interbase 6, Firebird 1 and newer database servers.

`TIBConnection` by default requires the Firebird/Interbase client library (e.g. `gds32.dll`, `libfbclient.so`, `fbclient.dll`, `fbembed.dll`) and its dependencies to be installed on the system. The bitness between library and your application must match: e.g. use 32 bit `fbclient` when developing a 32 bit application on 64 bit Linux.

On Windows, in accordance with the regular Windows way of loading DLLs, the library can also be in the executable directory. In fact, this directory is searched first, and might be a good option for distributing software to end users as it eliminates problems with incompatible DLL versions.

`TIBConnection` is based on FPC Interbase/Firebird code (`ibase60.inc`) that tries to load the client library. If you want to use Firebird embedded, make sure the embedded library is searched/loaded first. There are several ways to do this:

- Include `ibase60` in your uses clause, set `UseEmbeddedFirebird` to `true`
- On Windows, with FPC newer than 2.5.1, put `fbembed.dll` in your application directory
- On Windows, put the `fbembed.dll` in your application directory and rename it to `fbclient.dll`

Pre 2.5.1 versions of FPC did not try to load the `fbembed` library by default. See [FPC bug 17664](#) for more details.

An indication of which DLLs need to be installed on Windows (Firebird 2.5, differs between versions:

- `fbclient.dll` (or `fbembed.dll`)
- `firebird.msg`
- `ib_util.dll`
- `icudt30.dll`
- `icuin30.dll`
- `icuuc30.dll`
- `msvcp80.dll`
- `msvcr80.dll`

Please see your database documentation for details.

See also: `TSQLConnection` (??)

17.4.2 Method overview

Page	Property	Description
438	Create	Creates a <code>TIBConnection</code> object
438	CreateDB	Creates a database on disk
438	DropDB	Deletes a database from disk
439	GetDBDialect	Retrieves database dialect

17.4.3 Property overview

Page	Property	Access	Description
439	BlobSegmentSize	rw	Write this amount of bytes per BLOB segment
439	DatabaseName		Name of the database to connect to
440	Dialect	rws	Database dialect
440	KeepConnection		Keep open connection after first query
440	LoginPrompt		Switch for showing custom login prompt
441	OnLogin		Event triggered when a login prompt needs to be shown.
441	Params		Firebird/Interbase specific parameters

17.4.4 TIBConnection.Create

Synopsis: Creates a `TIBConnection` object

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

Visibility: `public`

Description: Creates a `TIBConnection` object

17.4.5 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 `Params` (??) (specifically, `PAGE_SIZE`) and `CharSet` (??) 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` (??), `TSQLConnection.UserName` (??), `TSQLConnection.Password` (??).

If creating a database using Firebird embedded, make sure the embedded library is loaded, the `HostName` (??) property is empty, and set the `TSQLConnection.UserName` (??) to e.g. 'SYSDBA'. See `TIBConnection`: Firebird/Interbase specific `TSQLConnection` (??) for details on loading the embedded database library.

See also: `Params` (??), `DropDB` (??), `TIBConnection` ([437](#))

17.4.6 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` ([437](#)).

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. `HostName` (??), `UserName` (??), `Password` (??).

When using Firebird embedded, make sure the embedded connection library is loaded, the `HostName` (??) property is empty, and set the `UserName` (??) to e.g. `'SYSDBA'`. See `TIBConnection: Firebird/Interbase specific TSQLConnection` (??) for more details on loading the embedded library.

See also: `CreateDB` (??), `HostName` (??), `UserName` (??), `Password` (??)

17.4.7 `TIBConnection.GetDBDialect`

Synopsis: Retrieves database dialect

Declaration: `function GetDBDialect : Integer`

Visibility: public

Description: Gets the Interbase/Firebird database dialect. You should take account of dialect when sending queries to the database.

Note: the dialect for new Interbase/Firebird databases is 3; dialects 1 and 2 are only used in legacy environments. See your database documentation for more details.

See also: `Dialect` (??)

17.4.8 `TIBConnection.BlobSegmentSize`

Synopsis: Write this amount of bytes per BLOB segment

Declaration: `Property BlobSegmentSize : Word; deprecated;`

Visibility: public

Access: Read,Write

Description: **Deprecated** since FPC 2.7.1 revision 19659

When sending BLOBs to the database, the code writes them in segments.

Before FPC 2.7.1 revision 19659, these segments were 80 bytes and could be changed using `BlobSegmentSize`. Please set `BlobSegmentSize` to 65535 for better write performance.

In newer FPC versions, the `BlobSegmentSize` property is ignored and segments of 65535 bytes are always used.

17.4.9 `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.

17.4.10 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.

17.4.11 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.

17.4.12 TIBConnection.LoginPrompt

Synopsis: Switch for showing custom login prompt

Declaration: `Property LoginPrompt :`

Visibility: published

Access:

Description: If true, the `OnLogin (??)` event will fire, allowing you to handle supplying of credentials yourself.

See also: `OnLogin (??)`

17.4.13 TIBConnection.Params

Synopsis: Firebird/Interbase specific parameters

Declaration: `Property Params :`

Visibility: published

Access:

Description: `Params` is a `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: `TSQLConnection.Params` (??)

17.4.14 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` (??) property is `True`, after the `BeforeConnect` (??) event, but before the connection is actually established.

See also: `BeforeConnect` (??), `LoginPrompt` (??), `Open` (??), `TSQLConnection.OnLogin` (??)

17.5 TIBConnectionDef

17.5.1 Description

Child of `SQLConnectionDef` (??); 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: `SQLConnectionDef` (??)

17.5.2 Method overview

Page	Property	Description
442	<code>ConnectionClass</code>	Firebird/Interbase child of <code>SQLConnectionDef.ConnectionClass</code> (??)
442	<code>DefaultLibraryName</code>	
442	<code>Description</code>	Description for the Firebird/Interbase child of <code>SQLConnectionDef.ConnectionClass</code> (??)
442	<code>LoadFunction</code>	
442	<code>TypeName</code>	Firebird/Interbase child of <code>SQLConnectionDef.TypeName</code> (??)
442	<code>UnLoadFunction</code>	

17.5.3 TIBConnectionDef.TypeName

Synopsis: Firebird/Interbase child of SQLConnectionDef.TypeName (??)

Declaration: `class function TypeName; Override`

Visibility: default

See also: SQLConnectionDef.TypeName (??), TIBConnection ([437](#))

17.5.4 TIBConnectionDef.ConnectionClass

Synopsis: Firebird/Interbase child of SQLConnectionDef.ConnectionClass (??)

Declaration: `class function ConnectionClass; Override`

Visibility: default

See also: SQLConnectionDef.ConnectionClass (??), TIBConnection ([437](#))

17.5.5 TIBConnectionDef.Description

Synopsis: Description for the Firebird/Interbase child of SQLConnectionDef.ConnectionClass (??)

Declaration: `class function Description; Override`

Visibility: default

Description: The description identifies this `ConnectionDef` object as a Firebird/Interbase connection.

See also: SQLConnectionDef.Description (??), TIBConnection ([437](#))

17.5.6 TIBConnectionDef.DefaultLibraryName

Declaration: `class function DefaultLibraryName; Override`

Visibility: default

17.5.7 TIBConnectionDef.LoadFunction

Declaration: `class function LoadFunction; Override`

Visibility: default

17.5.8 TIBConnectionDef.UnLoadFunction

Declaration: `class function UnLoadFunction; Override`

Visibility: default

17.6 TIBCursor

17.6.1 Description

A cursor that keeps track of where you are in a Firebird/Interbase dataset. It is a descendent of TSQLCursor (??).

See also: TSQLCursor (??), TIBConnection ([437](#))

17.7 TIBTrans

17.7.1 Description

Firebird/Interbase database transaction object. Descendant of TSQLHandle (??).

See also: TSQLHandle (??), TIBConnection ([437](#))

Chapter 18

Reference for unit 'idea'

18.1 Used units

Table 18.1: Used units by unit 'idea'

Name	Page
Classes	??
System	??
sysutils	??

18.2 Overview

Besides some low level IDEA encryption routines, the IDEA unit also offers 2 streams which offer on-the-fly encryption or decryption: there are 2 stream objects: A write-only encryption stream which encrypts anything that is written to it, and a decryption stream which decrypts anything that is read from it.

18.3 Constants, types and variables

18.3.1 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

18.3.2 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` (446) or `DeKeyIDEA` (445) 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.

18.4 Procedures and functions

18.4.1 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` (446), `DeKeyIdea` (445), `TIDEAEncryptStream` (448), `TIDEADecryptStream` (446)

18.4.2 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` (446), `CipherIdea` (445)

18.4.3 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` (445), `CipherIdea` (445)

18.5 EIDEAError

18.5.1 Description

`EIDEAError` is used to signal errors in the IDEA encryption decryption streams.

18.6 TIDEADeCryptStream

18.6.1 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` (445) 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` (448) encryption stream.

See also: `TIDEAEncryptStream` (448), `TIDEAStream.Create` (??), `CipherIDEA` (445)

18.6.2 Method overview

Page	Property	Description
447	Create	Constructor to create a new <code>TIDEADeCryptStream</code> instance
447	Read	Reads data from the stream, decrypting it as needed
447	Seek	Set position on the stream

18.6.3 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 (445), which is then passed on to the inherited constructor `TIDEAStream.Create` (??). It is an easy-access function which introduces no new functionality.

The string is truncated at the maximum length of the `TIdeaCryptKey` (445) structure, so it makes no sense to provide a string with length longer than this structure.

See also: `TIdeaCryptKey` (445), `TIDEAStream.Create` (??), `TIDEAEncryptStream.Create` (??)

18.6.4 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` (??), `Seek` (??), `TIDEAEncryptStream` (448)

18.6.5 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` (446) exception is raised if the stream does not allow the requested seek operation.

See also: `Read` (??)

18.7 TIDEAEncryptStream

18.7.1 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` (445) 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` (446) decryption stream.

See also: `TIDEADecryptStream` (446), `TIDEAStream.Create` (??), `CipherIDEA` (445)

18.7.2 Method overview

Page	Property	Description
448	Create	Constructor to create a new <code>TIDEAEncryptStream</code> instance
448	Destroy	Flush data buffers and free the stream instance.
449	Flush	Write remaining bytes from the stream
449	Seek	Set stream position
449	Write	Write bytes to the stream to be encrypted

18.7.3 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 the string `AKey` to compute the encryption key (445), which is then passed on to the inherited constructor `TIDEAStream.Create` (??). It is an easy-access function which introduces no new functionality.

The string is truncated at the maximum length of the `TIdeaCryptKey` (445) structure, so it makes no sense to provide a string with length longer than this structure.

See also: `TIdeaCryptKey` (445), `TIDEAStream.Create` (??), `TIDEADeCryptStream.Create` (??)

18.7.4 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` (??)

18.7.5 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` (??)

18.7.6 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` (446) will be raised unless called with 0 and `soFromCurrent` as arguments.

See also: `Write` (??), `EIDEAError` (446)

18.7.7 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` (??)

18.8 TIDEAStream

18.8.1 Description

Do not create instances of `TIDEAStream` directly. It implements no useful functionality: it serves as a common ancestor of the `TIDEAEncryptStream` (448) and `TIDEADeCryptStream` (446), 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` (448), `TIDEADeCryptStream` (446)

18.8.2 Method overview

Page	Property	Description
450	Create	Creates a new instance of the <code>TIDEASStream</code> class

18.8.3 Property overview

Page	Property	Access	Description
450	Key	r	Key used when encrypting/decrypting

18.8.4 TIDEASStream.Create

Synopsis: Creates a new instance of the `TIDEASStream` 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` ([448](#)), `TIDEADeCryptStream` ([446](#))

18.8.5 TIDEASStream.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` ([445](#))

Chapter 19

Reference for unit 'inicol'

19.1 Used units

Table 19.1: Used units by unit 'inicol'

Name	Page
Classes	??
IniFiles	461
System	??
sysutils	??

19.2 Overview

`inicol` contains an implementation of `TCollection` and `TCollectionItem` descendents which cooperate to read and write the collection from and to a `.ini` file. It uses the `TCustomIniFile` ([461](#)) class for this.

19.3 Constants, types and variables

19.3.1 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.

19.4 EIniCol

19.4.1 Description

EIniCol is used to report error conditions in the load and save methods of TIniCollection (452).

19.5 TIniCollection

19.5.1 Description

TIniCollection is a collection (??) descendent which has the capability to write itself to an .ini file. It introduces some load and save mechanisms, which can be used to write all items in the collection to disk. The items should be descendents of the type TIniCollectionItem (456).

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 (??), TIniCollection.LoadFromIni (??), TIniCollection.SaveToIni (??), TIniCollection.SaveToFile (??)

19.5.2 Method overview

Page	Property	Description
452	Load	Loads the collection from the default filename.
454	LoadFromFile	Load collection from file.
454	LoadFromIni	Load collection from a file in .ini file format.
453	Save	Save the collection to the default filename.
453	SaveToFile	Save collection to a file in .ini file format
453	SaveToIni	Save the collection to a TCustomIniFile descendent

19.5.3 Property overview

Page	Property	Access	Description
455	FileName	rw	Filename of the collection
455	GlobalSection	rw	Name of the global section
454	Prefix	r	Prefix used in global section
455	SectionPrefix	r	Prefix string for section names

19.5.4 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 (??) property. It calls the LoadFromFile (??) 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 (??)`, `TIniCollection.LoadFromIni (??)`, `TIniCollection.Save (??)`, `FileName (??)`

19.5.5 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 (??)` property, using `GlobalSection (??)` as the section. It calls the `SaveToFile (??)` 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 (??)`, `TIniCollection.SaveToIni (??)`, `TIniCollection.Load (??)`, `FileName (??)`

19.5.6 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 (??)` property for the key. After this it calls the `SaveToIni (??)` method of all `TIniCollectionItem` (456) 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 (??)` property contains the value of `Section`, it will be used in the `Save (??)` method.

See also: `TIniCollectionItem.SaveToIni (??)`

19.5.7 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 (??)` 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 (??)`, `TIniCollection.LoadFromFile (??)`

19.5.8 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 (??)` method, passing the appropriate section name as found in the global section.

The description of the global section can be found in the `TIniCollection.SaveToIni (??)` method description.

See also: `TIniCollection.LoadFromFile (??)`, `TIniCollectionItem.LoadFromIni (??)`, `TIniCollection.SaveToIni (??)`

19.5.9 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 (??)` using this instance and `Section` as the parameters.

See also: `TIniCollection.LoadFromIni (??)`, `TIniCollection.Load (??)`, `TIniCollection.SaveToIni (??)`, `TIniCollection.SaveToFile (??)`

19.5.10 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` (??), `TIniCollection.GlobalSection` (??)

19.5.11 `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` (??) property to return the exact section name. It can be empty.

See also: `TIniCollection.Section` (??), `TIniCollection.GlobalSection` (??)

19.5.12 `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` (??) or `SaveToFile` (??) operation. It is used in the `Load` (??) or `Save` (??) calls.

See also: `Save` (??), `LoadFromFile` (??), `SaveToFile` (??), `Load` (??)

19.5.13 `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` (??) or `SaveToIni` (??) calls. It's used in the `Load` (??) or `Save` (??) calls.

See also: `Save` (??), `LoadFromFile` (??), `SaveToFile` (??), `Load` (??)

19.6 TIniCollectionItem

19.6.1 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 (??) and LoadFromIni (??) methods should be overridden. They should implement the actual loading and saving. The loading and saving is always initiated by the methods in TIniCollection (452), TIniCollection.LoadFromIni (??) and TIniCollection.SaveToIni (??) respectively.

See also: TIniCollection (452), TIniCollectionItem.SaveToIni (??), TIniCollectionItem.LoadFromIni (??), TIniCollection.LoadFromIni (??), TIniCollection.SaveToIni (??)

19.6.2 Method overview

Page	Property	Description
457	LoadFromFile	Load item from a file
456	LoadFromIni	Method called when the item must be loaded
457	SaveToFile	Save item to a file
456	SaveToIni	Method called when the item must be saved

19.6.3 Property overview

Page	Property	Access	Description
457	SectionName	rw	Default section name

19.6.4 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 (??) 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 (??), LoadFromIni (??), SaveToFile (??), LoadFromFile (??)

19.6.5 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 (??)` 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 (??)`, `SaveToIni (??)`, `LoadFromFile (??)`, `SaveToFile (??)`

19.6.6 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 (??)` 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 (??)`, `LoadFromFile (??)`

19.6.7 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 (??)` to load the item from the indicated file in .ini format from the section `Section`.

Errors: None.

See also: `SaveToFile (??)`, `LoadFromIni (??)`

19.6.8 TIniCollectionItem.SectionName

Synopsis: Default section name

Declaration: `Property SectionName : string`

Visibility: public

Access: Read, Write

Description: `SectionName` is the section name under which the item will be saved or from which it should be read. The read/write functions should be overridden in descendents to determine a unique section name within the .ini file.

See also: `SaveToFile (??)`, `LoadFromIni (??)`

19.7 TNamedIniCollection

19.7.1 Description

TNamedIniCollection is the collection to go with the TNamedIniCollectionItem (459) item class. it provides some functions to look for items based on the UserData (??) or based on the Name (??).

See also: TNamedIniCollectionItem (459), IndexOfUserData (??), IndexOfName (??)

19.7.2 Method overview

Page	Property	Description
459	FindByName	Return the item based on its name
459	FindByUserData	Return the item based on its UserData
458	IndexOfName	Search for an item, based on its name, and return its position
458	IndexOfUserData	Search for an item based on it's UserData property

19.7.3 Property overview

Page	Property	Access	Description
459	NamedItems	rw	Indexed access to the TNamedIniCollectionItem items

19.7.4 TNamedIniCollection.IndexOfUserData

Synopsis: Search for an item based on it's UserData property

Declaration: `function IndexOfUserData(UserData: TObject) : Integer`

Visibility: public

Description: IndexOfUserData searches the list of items and returns the index of the item which has UserData in its UserData (??) 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 (??), TNamedIniCollectionItem.UserData (??)

19.7.5 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 (??), TNamedIniCollectionItem.Name (??)

19.7.6 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 (??)` 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 (??)`, `FindByUserData (??)`

19.7.7 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 (??)` property value matches the `UserData` parameter. If no item is found, `Nil` is returned.

Errors: If no item is found, `Nil` is returned.

19.7.8 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` (459) items. The index is zero based.

See also: `TNamedIniCollectionItem` (459)

19.8 TNamedIniCollectionItem

19.8.1 Description

`TNamedIniCollectionItem` is a `TIniCollectionItem` (456) 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` (456)

19.8.2 Property overview

Page	Property	Access	Description
460	<code>Name</code>	rw	Name of the item
460	<code>UserData</code>	rw	User-defined data

19.8.3 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`.

19.8.4 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` (??)

Chapter 20

Reference for unit 'IniFiles'

20.1 Used units

Table 20.1: Used units by unit 'IniFiles'

Name	Page
Classes	??
contnrs	106
System	??
sysutils	??

20.2 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 ([461](#)) and is implemented in TIniFile ([472](#)) and TMemIniFile ([481](#)). The API presented by these components is Delphi compatible.

20.3 TCustomIniFile

20.3.1 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 ([472](#)) or TMemIniFile ([481](#)).

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 ([472](#)), TMemIniFile ([481](#))

20.3.2 Method overview

Page	Property	Description
462	Create	Instantiate a new instance of TCustomIniFile.
469	DeleteKey	Delete a key from a section
463	Destroy	Remove the TCustomIniFile instance from memory
469	EraseSection	Clear a section
466	ReadBinaryStream	Read binary data
464	ReadBool	
465	ReadDate	Read a date value
465	ReadDateTime	Read a Date/Time value
466	ReadFloat	Read a floating point value
464	ReadInteger	Read an integer value from the file
468	ReadSection	Read the key names in a section
468	ReadSections	Read the list of sections
469	ReadSectionValues	Read names and values of a section
463	ReadString	Read a string valued key
466	ReadTime	Read a time value
463	SectionExists	Check if a section exists.
469	UpdateFile	Update the file on disk
470	ValueExists	Check if a value exists
468	WriteBinaryStream	Write binary data
465	WriteBool	Write boolean value
467	WriteDate	Write date value
467	WriteDateTime	Write date/time value
467	WriteFloat	Write a floating-point value
464	WriteInteger	Write an integer value
464	WriteString	Write a string value
467	WriteTime	Write time value

20.3.3 Property overview

Page	Property	Access	Description
470	CaseSensitive	rw	Are key and section names case sensitive
470	EscapeLineFeeds	r	Should linefeeds be escaped ?
470	FileName	r	Name of the .ini file
471	StripQuotes	rw	Should quotes be stripped from string values

20.3.4 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` (??)

20.3.5 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` ([461](#))

20.3.6 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` (??)

20.3.7 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` (??), `ReadInteger` (??), `ReadBool` (??), `ReadDate` (??), `ReadDateTime` (??), `ReadTime` (??), `ReadFloat` (??), `ReadBinaryStream` (??)

20.3.8 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.

See also: `ReadString` (??), `WriteInteger` (??), `WriteBool` (??), `WriteDate` (??), `WriteDateTime` (??), `WriteTime` (??), `WriteFloat` (??), `WriteBinaryStream` (??)

20.3.9 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` (??), `ReadString` (??), `ReadBool` (??), `ReadDate` (??), `ReadDateTime` (??), `ReadTime` (??), `ReadFloat` (??), `ReadBinaryStream` (??)

20.3.10 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` (??), `WriteString` (??), `WriteBool` (??), `WriteDate` (??), `WriteDateTime` (??), `WriteTime` (??), `WriteFloat` (??), `WriteBinaryStream` (??)

20.3.11 TCustomIniFile.ReadBool

Synopsis:

Declaration: `function ReadBool(const Section: string; const Ident: string;
Default: Boolean) : Boolean; Virtual`

Visibility: public

Description: `ReadString` reads the key `Ident` in section `Section`, and returns the value as a boolean (valid values are 0 and 1). If the specified key or section do not exist, then the value in `Default` is returned. If the key exists, but contains an invalid integer value, `False` is also returned.

See also: `WriteBool` (??), `ReadInteger` (??), `ReadString` (??), `ReadDate` (??), `ReadDateTime` (??), `ReadTime` (??), `ReadFloat` (??), `ReadBinaryStream` (??)

20.3.12 TCustomIniFile.WriteBool

Synopsis: Write boolean value

Declaration: `procedure WriteBool(const Section: string; const Ident: string;
Value: Boolean); Virtual`

Visibility: public

Description: `WriteBool` writes the boolean `Value` with the name `Ident` to the section `Section`, overwriting any previous value that may exist there. The section will be created if it does not exist.

See also: `ReadBool` (??), `WriteInteger` (??), `WriteString` (??), `WriteDate` (??), `WriteDateTime` (??), `WriteTime` (??), `WriteFloat` (??), `WriteBinaryStream` (??)

20.3.13 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` (??), `ReadInteger` (??), `ReadBool` (??), `ReadString` (??), `ReadDateTime` (??), `ReadTime` (??), `ReadFloat` (??), `ReadBinaryStream` (??)

20.3.14 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` (??), `ReadInteger` (??), `ReadBool` (??), `ReadDate` (??), `ReadString` (??), `ReadTime` (??), `ReadFloat` (??), `ReadBinaryStream` (??)

20.3.15 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` (??), `ReadInteger` (??), `ReadBool` (??), `ReadDate` (??), `ReadDateTime` (??), `ReadTime` (??), `ReadString` (??), `ReadBinaryStream` (??)

20.3.16 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` (??), `ReadInteger` (??), `ReadBool` (??), `ReadDate` (??), `ReadDateTime` (??), `ReadString` (??), `ReadFloat` (??), `ReadBinaryStream` (??)

20.3.17 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` (??), `ReadInteger` (??), `ReadBool` (??), `ReadDate` (??), `ReadDateTime` (??), `ReadTime` (??), `ReadFloat` (??), `ReadString` (??)

20.3.18 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` (??), `WriteInteger` (??), `WriteBool` (??), `WriteString` (??), `WriteDateTime` (??), `WriteTime` (??), `WriteFloat` (??), `WriteBinaryStream` (??)

20.3.19 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` (??), `WriteInteger` (??), `WriteBool` (??), `WriteDate` (??), `WriteString` (??), `WriteTime` (??), `WriteFloat` (??), `WriteBinaryStream` (??)

20.3.20 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` (??), `WriteInteger` (??), `WriteBool` (??), `WriteDate` (??), `WriteDateTime` (??), `WriteTime` (??), `WriteString` (??), `WriteBinaryStream` (??)

20.3.21 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` (??), `WriteInteger` (??), `WriteBool` (??), `WriteDate` (??), `WriteDateTime` (??), `WriteString` (??), `WriteFloat` (??), `WriteBinaryStream` (??)

20.3.22 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` (??), `WriteInteger` (??), `WriteBool` (??), `WriteDate` (??), `WriteDateTime` (??), `WriteTime` (??), `WriteFloat` (??), `WriteString` (??)

20.3.23 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` (??) method should be used.

See also: `ReadSections` (??), `SectionExists` (??), `ReadSectionValues` (??)

20.3.24 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` (??), `ReadSectionValues` (??), `ReadSection` (??)

20.3.25 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 (??)` can be used.

See also: `SectionExists (??)`, `ReadSections (??)`, `ReadSection (??)`

20.3.26 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 (??)`, `ReadSections (??)`, `DeleteKey (??)`

20.3.27 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 (??)`

20.3.28 TCustomIniFile.UpdateFile

Synopsis: Update the file on disk

Declaration: `procedure UpdateFile; Virtual; Abstract`

Visibility: public

Description: `UpdateFile` writes the in-memory image of the ini-file to disk. To speed up operation of the inifile class, the whole ini-file is read into memory when the class is created, and all operations are performed in-memory. If `CacheUpdates` is set to `True`, any changes to the inifile are only in memory, until they are committed to disk with a call to `UpdateFile`. If `CacheUpdates` is set to `False`, then all operations which cause a change in the .ini file will immediately be committed to disk with a call to `UpdateFile`. Since the whole file is written to disk, this may have serious impact on performance.

See also: `CacheUpdates (??)`

20.3.29 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` (??)

20.3.30 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` (??)

20.3.31 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` (??), as the value of this property must be specified when the `TCustomIniFile` instance is created.

By default, `EscapeLineFeeds` is `False`.

See also: `Create` (??), `CaseSensitive` (??)

20.3.32 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` (??)

20.3.3 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).

20.4 THashedStringList

20.4.1 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` (??) and `IndexOfName` (??) functions make use of these hash values to quicklier locate a value.

See also: `IndexOf` (??), `IndexOfName` (??)

20.4.2 Method overview

Page	Property	Description
471	<code>Destroy</code>	Clean up instance
471	<code>IndexOf</code>	Returns the index of a string in the list of strings
472	<code>IndexOfName</code>	Return the index of a name in the list of name=value pairs

20.4.3 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` (??)

20.4.4 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` (??)

20.4.5 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 (??)`

20.5 TIniFile

20.5.1 Description

`TIniFile` is an implementation of `TCustomIniFile` (461) which does the same as `TMemIniFile` (481), namely it reads the whole file into memory. Unlike `TMemIniFile` it does not cache updates in memory, but immediatly writes any changes to disk.

`TIniFile` introduces no new methods, it just implements the abstract methods introduced in `TCustomIniFile`

See also: `TCustomIniFile` (461), `TMemIniFile` (481)

20.5.2 Method overview

Page	Property	Description
472	Create	Create a new instance of <code>TIniFile</code>
475	DeleteKey	Delete key
473	Destroy	Remove the <code>TIniFile</code> instance from memory
475	EraseSection	
474	ReadSection	Read the key names in a section
474	ReadSectionRaw	Read raw section
474	ReadSections	Read section names
474	ReadSectionValues	
473	ReadString	Read a string
475	UpdateFile	Update the file on disk
473	WriteString	Write string to file

20.5.3 Property overview

Page	Property	Access	Description
476	CacheUpdates	rw	Should changes be kept in memory
475	Stream	r	Stream from which ini file was read

20.5.4 TIniFile.Create

Synopsis: Create a new instance of `TIniFile`

Declaration: `constructor Create(const AFileName: string; AEscapeLineFeeds: Boolean)`
`; Override`
`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` ([461](#)), `TMemIniFile` ([481](#))

20.5.5 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` ([??](#)), `CacheUpdates` ([??](#))

20.5.6 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` ([??](#)) abstract method by looking at the in-memory copy of the ini file and returning the string found there.

See also: `TCustomIniFile.ReadString` ([??](#))

20.5.7 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` ([??](#)) abstract method by writing the string to the in-memory copy of the ini file. If `CacheUpdates` ([??](#)) property is `False`, then the whole file is immediatly written to disk as well.

Errors: If an error happens when the file is written to disk, an exception will be raised.

20.5.8 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` (??)

See also: `TCustomIniFile.ReadSection` (??), `TIniFile.ReadSectionRaw` (??)

20.5.9 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` (??), `TCustomIniFile.ReadSection` (??)

20.5.10 TIniFile.ReadSections

Synopsis: Read section names

Declaration: `procedure ReadSections(Strings: TStrings); Override`

Visibility: public

Description: `ReadSections` is the implementation of `TCustomIniFile.ReadSections` (??). It operates on the in-memory copy of the inifile, and places all section names in `Strings`.

See also: `TIniFile.ReadSection` (??), `TCustomIniFile.ReadSections` (??), `TIniFile.ReadSectionValues` (??)

20.5.11 TIniFile.ReadSectionValues

Synopsis:

Declaration: `procedure ReadSectionValues(const Section: string; Strings: TStrings)
; Override`

Visibility: public

Description: `ReadSectionValues` is the implementation of `TCustomIniFile.ReadSectionValues` (??). It operates on the in-memory copy of the inifile, and places all key names from `Section` together with their values in `Strings`.

See also: `TIniFile.ReadSection` (??), `TCustomIniFile.ReadSectionValues` (??), `TIniFile.ReadSections` (??)

20.5.12 TIniFile.EraseSection

Synopsis:

Declaration: `procedure EraseSection(const Section: string); Override`

Visibility: `public`

Description: `EraseSection` deletes the section `Section` from memory, if `CacheUpdates (??)` is `False`, then the file is immediatly updated on disk. This method is the implementation of the abstract `TCustomIniFile.EraseSection (??)` method.

See also: `TCustomIniFile.EraseSection (??)`, `TIniFile.ReadSection (??)`, `TIniFile.ReadSections (??)`

20.5.13 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 (??)` is `False`, then the file is immediatly updated on disk.

See also: `CacheUpdates (??)`

20.5.14 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 (??)`

20.5.15 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 (??)` method will use this stream to write changes to.

See also: `Create (??)`, `UpdateFile (??)`

20.5.16 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` (??). 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` (??)

20.6 TIniFileKey

20.6.1 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` (472) class.

See also: `TIniFile` (472)

20.6.2 Method overview

Page	Property	Description
476	<code>Create</code>	Create a new instance of <code>TIniFileKey</code>

20.6.3 Property overview

Page	Property	Access	Description
477	<code>Ident</code>	<code>rw</code>	Key name
477	<code>Value</code>	<code>rw</code>	Key value

20.6.4 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` (??) with `AIdent` and `Value` (??) with `AValue`.

See also: `Ident` (??), `Value` (??)

20.6.5 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` (??)

20.6.6 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` (??)

20.7 TIniFileKeyList

20.7.1 Description

`TIniFileKeyList` maintains a list of `TIniFileKey` (476) instances on behalf of the `TIniFileSection` (478) class. It stores the keys of one section of the .ini files.

See also: `TIniFileKey` (476), `TIniFileSection` (478)

20.7.2 Method overview

Page	Property	Description
478	<code>Clear</code>	Clear the list
477	<code>Destroy</code>	Free the instance

20.7.3 Property overview

Page	Property	Access	Description
478	<code>Items</code>	<code>r</code>	Indexed access to <code>TIniFileKey</code> items in the list

20.7.4 TIniFileKeyList.Destroy

Synopsis: Free the instance

Declaration: `destructor Destroy; Override`

Visibility: public

Description: `Destroy` clears up the list using `Clear` (??) and then calls the inherited `destroy`.

See also: `Clear` (??)

20.7.5 TIniFileKeyList.Clear

Synopsis: Clear the list

Declaration: `procedure Clear; Override`

Visibility: `public`

Description: `Clear` removes all `TIniFileKey` (476) instances from the list, and frees the instances.

See also: `TIniFileKey` (476)

20.7.6 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` (476) items in the list. The index is zero-based and runs from 0 to `Count-1`.

See also: `TIniFileKey` (476)

20.8 TIniFileSection

20.8.1 Description

`TIniFileSection` is a class which represents a section in the `.ini`, and is used internally by the `TIniFile` (472) class (one instance of `TIniFileSection` is created for each section in the file by the `TIniFileSectionList` (480) list). The name of the section is stored in the `Name` (??) property, and the key/value pairs in this section are available in the `KeyList` (??) property.

See also: `TIniFileKeyList` (477), `TIniFile` (472), `TIniFileSectionList` (480)

20.8.2 Method overview

Page	Property	Description
479	Create	Create a new section object
479	Destroy	Free the section object from memory
479	Empty	Is the section empty

20.8.3 Property overview

Page	Property	Access	Description
480	KeyList	r	List of key/value pairs in this section
479	Name	r	Name of the section

20.8.4 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.

20.8.5 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` (477) instance to keep all the key/value pairs for this section.

See also: `TIniFileKeyList` (477)

20.8.6 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` (??), `TIniFileKeyList` (477)

20.8.7 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` (??)

20.8.8 TIniFileSection.KeyList

Synopsis: List of key/value pairs in this section

Declaration: `Property KeyList : TIniFileKeyList`

Visibility: `public`

Access: `Read`

Description: `KeyList` is the `TIniFileKeyList` (477) instance that is used by the `TIniFileSection` to keep the key/value pairs of the section.

See also: `TIniFileSection.Name` (??), `TIniFileKeyList` (477)

20.9 TIniFileSectionList

20.9.1 Description

`TIniFileSectionList` maintains a list of `TIniFileSection` (478) instances, one for each section in an .ini file. `TIniFileSectionList` is used internally by the `TIniFile` (472) class to represent the sections in the file.

See also: `TIniFileSection` (478), `TIniFile` (472)

20.9.2 Method overview

Page	Property	Description
480	<code>Clear</code>	Clear the list
480	<code>Destroy</code>	Free the object from memory

20.9.3 Property overview

Page	Property	Access	Description
481	<code>Items</code>	<code>r</code>	Indexed access to all the section objects in the list

20.9.4 TIniFileSectionList.Destroy

Synopsis: Free the object from memory

Declaration: `destructor Destroy; Override`

Visibility: `public`

Description: `Destroy` calls `Clear` (??) to clear the section list and the calls the inherited `Destroy`

See also: `Clear` (??)

20.9.5 TIniFileSectionList.Clear

Synopsis: Clear the list

Declaration: `procedure Clear; Override`

Visibility: `public`

Description: `Clear` removes all `TIniFileSection` (478) items from the list, and frees the items it removes from the list.

See also: `TIniFileSection` (478), `TIniFileSectionList.Items` (??)

20.9.6 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` (478), `TIniFileSectionList.Clear` (??)

20.10 TMemIniFile

20.10.1 Description

`TMemIniFile` is a simple descendent of `TIniFile` (472) 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` (472), `TCustomIniFile` (461), `CacheUpdates` (??)

20.10.2 Method overview

Page	Property	Description
482	<code>Clear</code>	Clear the data
481	<code>Create</code>	Create a new instance of <code>TMemIniFile</code>
482	<code>GetStrings</code>	Get contents of ini file as stringlist
482	<code>Rename</code>	Rename the ini file
482	<code>SetStrings</code>	Set data from a stringlist

20.10.3 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` (??), and sets the `CacheUpdates` (??) to `True` so updates will be kept in memory till they are explicitly written to disk.

See also: `TIniFile.Create` (??), `CacheUpdates` (??)

20.10.4 TMemIniFile.Clear

Synopsis: Clear the data

Declaration: `procedure Clear`

Visibility: `public`

Description: `Clear` removes all sections and key/value pairs from memory. If `CacheUpdates (??)` is set to `False` then the file on disk will immediately be emptied.

See also: `SetStrings (??)`, `GetStrings (??)`

20.10.5 TMemIniFile.GetStrings

Synopsis: Get contents of ini file as stringlist

Declaration: `procedure GetStrings(List: TStringList)`

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 (??)`, which can be used to copy data between 2 in-memory ini files.

See also: `SetStrings (??)`, `Clear (??)`

20.10.6 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 (??)`

20.10.7 TMemIniFile.SetStrings

Synopsis: Set data from a stringlist

Declaration: `procedure SetStrings(List: TStringList)`

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 (??)`. The two calls combined can be used to copy data between 2 in-memory ini files.

See also: `GetStrings (??)`, `Clear (??)`

Chapter 21

Reference for unit 'iostream'

21.1 Used units

Table 21.1: Used units by unit 'iostream'

Name	Page
Classes	??
System	??

21.2 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`).

21.3 Constants, types and variables

21.3.1 Types

`TIOSType = (iosInput, iosOutPut, iosError)`

Table 21.2: Enumeration values for type `TIOSType`

Value	Explanation
<code>iosError</code>	The stream can be used to write to standard diagnostic output
<code>iosInput</code>	The stream can be used to read from standard input
<code>iosOutPut</code>	The stream can be used to write to standard output

`TIOSType` is passed to the `Create` (??) constructor of `TIOStream` (484), it determines what kind of stream is created.

21.4 EIOStreamError

21.4.1 Description

Error thrown in case of an invalid operation on a TIOStream ([484](#)).

21.5 TIOStream

21.5.1 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 ([483](#)) 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 (??) behaviour based on this position.

See also: TIOSType ([483](#)), THandleStream (??)

21.5.2 Method overview

Page	Property	Description
484	Create	Construct a new instance of TIOStream (484)
484	Read	Read data from the stream.
485	Seek	Set the stream position
485	Write	Write data to the stream

21.5.3 TIOStream.Create

Synopsis: Construct a new instance of TIOStream ([484](#))

Declaration: `constructor Create(aIOSType: TIOSType)`

Visibility: public

Description: Create creates a new instance of TIOStream ([484](#)), 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 (??), TIOStream.Write (??)

21.5.4 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 ([484](#)) exception. If the stream can be read, it calls the inherited Read to actually read the data.

Errors: An `EIOStreamError` exception is raised if the stream does not allow reading.

See also: `TIOSType` (483), `TIOStream.Write` (??)

21.5.5 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` (484) exception. If the stream can be written to, it calls the inherited `Write` to actually read the data.

Errors: An `EIOStreamError` exception is raised if the stream does not allow writing.

See also: `TIOSType` (483), `TIOStream.Read` (??)

21.5.6 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` (484) exception is raised if the stream does not allow the requested seek operation.

See also: `EIOStreamError` (484)

Chapter 22

Reference for unit 'libtar'

22.1 Used units

Table 22.1: Used units by unit 'libtar'

Name	Page
BaseUnix	??
Classes	??
System	??
sysutils	??
Unix	??
UnixType	??
Windows	??

22.2 Overview

The `libtar` units provides 2 classes to read and write `.tar` archives: `TTarArchive` ([490](#)) class can be used to read a tar file, and the `TTarWriter` ([492](#)) class can be used to write a tar file. The unit was implemented originally by Stefan Heymann.

22.3 Constants, types and variables

22.3.1 Constants

`ALL_PERMISSIONS` = [`tpReadByOwner`, `tpWriteByOwner`, `tpExecuteByOwner`, `tpReadByGroup`, `tpWriteByGroup`, `tpExecuteByGroup`, `tpReadByOther`, `tpWriteByOther`, `tpExecuteByOther`]

`ALL_PERMISSIONS` is a set constant containing all possible permissions (read/write/execute, for all groups of users) for an archive entry.

`EXECUTE_PERMISSIONS` = [`tpExecuteByOwner`, `tpExecuteByGroup`, `tpExecuteByOther`]

`WRITE_PERMISSIONS` is a set constant containing all possible execute permissions set for an archive entry.

```
FILETYPE_NAME : Array[TFileType] of = ('Regular', 'Link', 'Symbolic Link', 'Char Fi
```

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.

22.3.2 Types

```
TFileType = (ftNormal, ftLink, ftSymbolicLink, ftCharacter, ftBlock,
             ftDirectory, ftFifo, ftContiguous, ftDumpDir, ftMultiVolume,
             ftVolumeHeader)
```

Table 22.2: Enumeration values for type TFileType

Value	Explanation
ftBlock	Block device file
ftCharacter	Character device file
ftContiguous	Contiguous file
ftDirectory	Directory
ftDumpDir	List of files
ftFifo	FIFO file
ftLink	Hard link
ftMultiVolume	Multi-volume file part
ftNormal	Normal file
ftSymbolicLink	Symbolic link
ftVolumeHeader	Volume header, can appear only as first entry in the archive

TFileType describes the file type of a file in the archive. It is used in the FileType field of the TTarDirRec (488) record.

```
TTarDirRec = record
  Name : AnsiString;
  Size : Int64;
  DateTime : TDateTime;
  Permissions : TTarPermissions;
  FileType : TFileType;
  LinkName : AnsiString;
  UID : Integer;
  GID : Integer;
  UserName : AnsiString;
  GroupName : AnsiString;
  ChecksumOK : Boolean;
```



```

Mode : TTarModes;
Magic : AnsiString;
MajorDevNo : Integer;
MinorDevNo : Integer;
FilePos : Int64;
end

```

TTarDirRec describes an entry in the tar archive. It is similar to a directory entry as in TSearchRec (??), and is returned by the TTarArchive.FindNext (??) call.

```
TTarMode = (tmSetUid, tmSetGid, tmSaveText)
```

Table 22.3: 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 (488) record.

```
TTarModes = Set of TTarMode
```

TTarModes denotes the full set of permission bits for the file in the field Mode field of the TTarDirRec (488) record.

```

TTarPermission = (tpReadByOwner, tpWriteByOwner, tpExecuteByOwner,
                  tpReadByGroup, tpWriteByGroup, tpExecuteByGroup,
                  tpReadByOther, tpWriteByOther, tpExecuteByOther)

```

Table 22.4: 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 (488) record.

22.4 Procedures and functions

22.4.1 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` ([488](#))

22.4.2 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.

22.4.3 FileTimeGMT

Synopsis: Extract filetype

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` ([488](#))

22.4.4 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` ([488](#))

22.5 TTarArchive

22.5.1 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` ([492](#)), `FindNext` ([??](#))

22.5.2 Method overview

Page	Property	Description
490	<code>Create</code>	Create a new instance of the archive
490	<code>Destroy</code>	Destroy <code>TTarArchive</code> instance
491	<code>FindNext</code>	Find next archive entry
491	<code>GetFilePos</code>	Return current archive position
491	<code>ReadFile</code>	Read a file from the archive
490	<code>Reset</code>	Reset archive
492	<code>SetFilePos</code>	Set position in archive

22.5.3 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` ([??](#))

22.5.4 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` ([??](#))

22.5.5 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` (??)

22.5.6 `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` (??)

22.5.7 `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.

22.5.8 `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` (??), `TTarArchive.Reset` (??)

22.5.9 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` (??), `TTarArchive.GetFilePos` (??)

22.6 TTarWriter

22.6.1 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` ([490](#))

22.6.2 Method overview

Page	Property	Description
494	<code>AddDir</code>	Add directory to archive
493	<code>AddFile</code>	Add a file to the archive
495	<code>AddLink</code>	Add hard link to archive
493	<code>AddStream</code>	Add stream contents to archive.
494	<code>AddString</code>	Add string as file data
494	<code>AddSymbolicLink</code>	Add a symbolic link to the archive
495	<code>AddVolumeHeader</code>	Add volume header entry
492	<code>Create</code>	Create a new archive
493	<code>Destroy</code>	Close archive and clean up <code>TTarWriter</code>
495	<code>Finalize</code>	Finalize the archive

22.6.3 Property overview

Page	Property	Access	Description
496	<code>GID</code>	rw	Archive entry group ID
496	<code>GroupName</code>	rw	Archive entry group name
497	<code>Magic</code>	rw	Archive entry Magic constant
497	<code>Mode</code>	rw	Archive entry mode
495	<code>Permissions</code>	rw	Archive entry permissions
496	<code>UID</code>	rw	Archive entry user ID
496	<code>UserName</code>	rw	Archive entry user name

22.6.4 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` (??)

22.6.5 `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` (??)

22.6.6 `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` (??), `TTarWriter.AddString` (??), `TTarWriter.AddLink` (??), `TTarWriter.AddSymbolicLink` (??), `TTarWriter.AddDir` (??), `TTarWriter.AddVolumeHeader` (??)

22.6.7 `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` (??), `TTarWriter.AddString` (??), `TTarWriter.AddLink` (??), `TTarWriter.AddSymbolicLink` (??), `TTarWriter.AddDir` (??), `TTarWriter.AddVolumeHeader` (??)

22.6.8 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` (??), `TTarWriter.AddStream` (??), `TTarWriter.AddLink` (??), `TTarWriter.AddSymbolicLink` (??), `TTarWriter.AddDir` (??), `TTarWriter.AddVolumeHeader` (??)

22.6.9 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` (??), `TTarWriter.AddStream` (??), `TTarWriter.AddLink` (??), `TTarWriter.AddSymbolicLink` (??), `TTarWriter.AddString` (??), `TTarWriter.AddVolumeHeader` (??)

22.6.10 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` (??), `TTarWriter.AddStream` (??), `TTarWriter.AddLink` (??), `TTarWriter.AddDir` (??), `TTarWriter.AddString` (??), `TTarWriter.AddVolumeHeader` (??)

22.6.11 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` (??), `TTarWriter.AddStream` (??), `TTarWriter.AddSymbolicLink` (??), `TTarWriter.AddDir` (??), `TTarWriter.AddString` (??), `TTarWriter.AddVolumeHeader` (??)

22.6.12 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` (??), `TTarWriter.AddStream` (??), `TTarWriter.AddSymbolicLink` (??), `TTarWriter.AddDir` (??), `TTarWriter.AddString` (??), `TTarWriter.AddLink` (??)

22.6.13 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` (??)

22.6.14 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` ([488](#))

22.6.15 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` ([488](#))

22.6.16 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` ([488](#))

22.6.17 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` ([488](#))

22.6.18 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` ([488](#))

22.6.19 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` ([488](#))

22.6.20 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` ([488](#))

Chapter 23

Reference for unit 'mssqlconn'

23.1 Used units

Table 23.1: Used units by unit 'mssqlconn'

Name	Page
BufDataset	??
Classes	??
db	218
dblib	??
sqldb	??
System	??
sysutils	??

23.2 Overview

Connector to Microsoft SQL Server databases. Needs FreeTDS dblib library.

23.3 Constants, types and variables

23.3.1 Types

`TClientCharset = (ccNone, ccUTF8, ccISO88591, ccUnknown)`

Table 23.2: Enumeration values for type TClientCharset

Value	Explanation
ccISO88591	
ccNone	
ccUnknown	
ccUTF8	

23.3.2 Variables

DBLibLibraryName : string = DBLIBDLL

23.4 EMSSQLDatabaseError

23.4.1 Description

Sybase/MS SQL Server specific error

23.5 TMSSQLConnection

23.5.1 Description

Connector to Microsoft SQL Server databases.

Requirements:

MS SQL Server Client Library is required (ntwdblib.dll)

- or -

FreeTDS (dblib.dll)

Older FreeTDS libraries may require freetds.conf: (<http://www.freetds.org/userguide/freetdsconf.htm>)

[global]

tds version = 7.1

client charset = UTF-8

port = 1433 or instance = ... (optional)

dump file = freetds.log (optional)

text size = 2147483647 (optional)

Known problems:

- CHAR/VARCHAR data truncated to column length when encoding to UTF-8 (use NCHAR/NVARCHAR instead or CAST char/varchar to nchar/nvarchar)

- Multiple result sets (MARS) are not supported (for example when SP returns more than 1 result set only 1st is processed)

- DB-Library error 10038 "Results Pending": set TSQLQuery.PacketRecords=-1 to fetch all pending rows

- BLOB data (IMAGE/TEXT columns) larger than 16MB are truncated to 16MB: (set TMSSQL-Connection.Params: 'TEXTSIZE=2147483647' or execute 'SET TEXTSIZE 2147483647')

23.5.2 Method overview

Page	Property	Description
500	Create	

23.5.3 Property overview

Page	Property	Access	Description
501	CharSet		
501	Connected		
501	DatabaseName		
501	HostName		Host and optionally port or instance
502	KeepConnection		
502	LoginPrompt		
502	OnLogin		
502	Params		
500	Password		
501	Role		
500	Transaction		
500	UserName		

23.5.4 TMSSQLConnection.Create

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

Visibility: `public`

23.5.5 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).

23.5.6 TMSSQLConnection.Transaction

Declaration: `Property Transaction :`

Visibility: `published`

Access:

23.5.7 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).

23.5.8 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.

23.5.9 TMSSQLConnection.HostName

Synopsis: Host and optionally port or instance

Declaration: `Property HostName :`

Visibility: published

Access:

Description: `TMSSQLConnection` specific: you can specify an instance or a port after the host name itself.

Instance should be specified with a backslash e.g.: 127.0.0.0.1\SQLEXPRESS. Port should be specified with a colon, e.g. BIGBADSERVER:1433

See <http://www.freetds.org/userguide/portoverride.htm>

23.5.10 TMSSQLConnection.Connected

Declaration: `Property Connected :`

Visibility: published

Access:

23.5.11 TMSSQLConnection.Role

Declaration: `Property Role :`

Visibility: published

Access:

23.5.12 TMSSQLConnection.DatabaseName

Declaration: `Property DatabaseName :`

Visibility: published

Access:

Description: `TMSSQLConnection` specific: the master database should always exist on a server.

23.5.13 TMSSQLConnection.KeepConnection

Declaration: `Property KeepConnection :`

Visibility: published

Access:

23.5.14 TMSSQLConnection.LoginPrompt

Declaration: `Property LoginPrompt :`

Visibility: published

Access:

23.5.15 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.

23.5.16 TMSSQLConnection.OnLogin

Declaration: `Property OnLogin :`

Visibility: published

Access:

23.6 TMSSQLConnectionDef

23.6.1 Method overview

Page	Property	Description
503	ConnectionClass	
503	Description	
502	TypeName	

23.6.2 TMSSQLConnectionDef.TypeName

Declaration: `class function TypeName; Override`

Visibility: default

23.6.3 TMSSQLConnectionDef.ConnectionClass

Declaration: `class function ConnectionClass; Override`

Visibility: `default`

23.6.4 TMSSQLConnectionDef.Description

Declaration: `class function Description; Override`

Visibility: `default`

23.7 TSybaseConnection

23.7.1 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)

23.7.2 Method overview

Page	Property	Description
503	Create	

23.7.3 TSybaseConnection.Create

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

Visibility: `public`

23.8 TSybaseConnectionDef

23.8.1 Method overview

Page	Property	Description
504	ConnectionClass	
504	Description	
504	TypeName	

23.8.2 TSybaseConnectionDef.TypeName

Declaration: `class function TypeName; Override`

Visibility: `default`

23.8.3 TSybaseConnectionDef.ConnectionClass

Declaration: `class function ConnectionClass; Override`

Visibility: `default`

23.8.4 TSybaseConnectionDef.Description

Declaration: `class function Description; Override`

Visibility: `default`

Chapter 24

Reference for unit 'Pipes'

24.1 Used units

Table 24.1: Used units by unit 'Pipes'

Name	Page
Classes	??
System	??
sysutils	??

24.2 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.

24.3 Constants, types and variables

24.3.1 Constants

`ENoSeekMsg = 'Cannot seek on pipes'`

Constant used in `EPipeSeek` ([506](#)) exception.

`EPipeMsg = 'Failed to create pipe.'`

Constant used in `EPipeCreation` ([506](#)) exception.

24.4 Procedures and functions

24.4.1 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` (506)

24.4.2 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` (505), and if that call is successful, a pair of streams is created: `InPipe` and `OutPipe`.

Errors: If no pipe handles could be created, an `EPipeCreation` (506) exception is raised.

See also: `CreatePipeHandles` (505), `TInputPipeStream` (506), `TOutputPipeStream` (508)

24.5 EPipeCreation

24.5.1 Description

Exception raised when an error occurred during the creation of a pipe pair.

24.6 EPipeError

24.6.1 Description

Exception raised when an invalid operation is performed on a pipe stream.

24.7 EPipeSeek

24.7.1 Description

Exception raised when an invalid seek operation is attempted on a pipe.

24.8 TInputPipeStream

24.8.1 Description

`TInputPipeStream` is created by the `CreatePipeStreams` (506) call to represent the reading end of a pipe. It is a `TStream` (??) descendent which does not allow writing, and which mimics the seek operation.

See also: TStream (??), CreatePipeStreams (506), TOutputPipeStream (508)

24.8.2 Method overview

Page	Property	Description
507	Destroy	
508	Read	Read data from the stream to a buffer.
507	Seek	Set the current position of the stream
507	Write	Write data to the stream.

24.8.3 Property overview

Page	Property	Access	Description
508	NumBytesAvailable	r	Number of bytes available for reading.

24.8.4 TInputPipeStream.Destroy

Declaration: `destructor Destroy; Override`

Visibility: `public`

24.8.5 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` (??), `Seek` (??)

24.8.6 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` (506) exception is raised if the stream does not allow the requested seek operation.

See also: `EPipeSeek` (506), `Seek` (??)

24.8.7 `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` (??), `Seek` (??)

24.8.8 `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` (??)

24.9 `TOutputPipeStream`

24.9.1 Description

`TOutputPipeStream` is created by the `CreatePipeStreams` (506) call to represent the writing end of a pipe. It is a `TStream` (??) descendent which does not allow reading.

See also: `TStream` (??), `CreatePipeStreams` (506), `TInputPipeStream` (506)

24.9.2 Method overview

Page	Property	Description
509	<code>Destroy</code>	
509	<code>Read</code>	Read data from the stream.
509	<code>Seek</code>	Sets the position in the stream

24.9.3 TOutputPipeStream.Destroy

Declaration: `destructor Destroy; Override`

Visibility: `public`

24.9.4 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` (506) exception is raised if this method is called.

24.9.5 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` (??)

Chapter 25

Reference for unit 'pooledmm'

25.1 Used units

Table 25.1: Used units by unit 'pooledmm'

Name	Page
Classes	??
System	??

25.2 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.

25.3 Constants, types and variables

25.3.1 Types

```
PPooledMemManagerItem = ^TPooledMemManagerItem
```

`PPooledMemManagerItem` is a pointer type, pointing to a `TPooledMemManagerItem` (511) item, used in a linked list.

```
TEnumItemsMethod = procedure(Item: Pointer) of object
```

`TEnumItemsMethod` is a prototype for the callback used in the `TNonFreePooledMemManager.EnumerateItems` (??) call. The parameter `Item` will be set to each of the pointers in the item list of `TNonFreePooledMemManager` (511).

```
TPooledMemManagerItem = record
  Next : PPooledMemManagerItem;
end
```

`TPooledMemManagerItem` is used internally by the `TPooledMemManager` (513) class to maintain the free list block. It simply points to the next free block.

25.4 TNonFreePooledMemManager

25.4.1 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 an exponential way, i.e. each time a new block of memory must be allocated, its size is the double of the last block. The first block will contain 8 items.

25.4.2 Method overview

Page	Property	Description
511	<code>Clear</code>	Clears the memory
511	<code>Create</code>	Creates a new instance of <code>TNonFreePooledMemManager</code>
512	<code>Destroy</code>	Removes the <code>TNonFreePooledMemManager</code> instance from memory
512	<code>EnumerateItems</code>	Enumerate all items in the list
512	<code>NewItem</code>	Return a pointer to a new memory block

25.4.3 Property overview

Page	Property	Access	Description
512	<code>ItemSize</code>	r	Size of an item in the list

25.4.4 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` (??) is valid after a call to `Clear`

See also: `NewItem` (??)

25.4.5 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` (??)

25.4.6 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` (??), `TNonFreePooledMemManager.Clear` (??)

25.4.7 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` (??). 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` (??)

25.4.8 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)

25.4.9 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` (??)

25.5 TPooledMemManager

25.5.1 Description

`TPooledMemManager` is a class which maintains a linked list of blocks, represented by the `TPooledMemManagerItem` (511) record. It should not be used directly, but should be descended from and the descendent should implement the actual memory manager.

See also: `TPooledMemManagerItem` (511)

25.5.2 Method overview

Page	Property	Description
513	<code>Clear</code>	Clears the list
513	<code>Create</code>	Creates a new instance of the <code>TPooledMemManager</code> class
514	<code>Destroy</code>	Removes an instance of <code>TPooledMemManager</code> class from memory

25.5.3 Property overview

Page	Property	Access	Description
515	<code>AllocatedCount</code>	r	Total number of allocated items in the list
514	<code>Count</code>	r	Number of items in the list
515	<code>FreeCount</code>	r	Number of free items in the list
515	<code>FreedCount</code>	r	Total number of freed items in the list.
514	<code>MaximumFreeCountRatio</code>	rw	Maximum ratio of free items over total items
514	<code>MinimumFreeCount</code>	rw	Minimum count of free items in the list

25.5.4 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` (??)

25.5.5 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` (??)

25.5.6 TPooledMemManager.Destroy

Synopsis: Removes an instance of `TPooledMemManager` class from memory

Declaration: `destructor Destroy; Override`

Visibility: `public`

Description: `Destroy` calls `Clear (??)` and then calls the inherited `destroy`.

`Destroy` should never be called directly. Instead `Free` should be used, or `FreeAndNil`

See also: `TPooledMemManager.Create (??)`

25.5.7 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 (??)`

25.5.8 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 (??)`

25.5.9 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 (??)`, `TPooledMemManager.AllocatedCount (??)`, `TPooledMemManager.FreedCount (??)`

25.5.10 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` (??), `TPooledMemManager.AllocatedCount` (??), `TPooledMemManager.FreedCount` (??)

25.5.11 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` (??), `TPooledMemManager.FreeCount` (??), `TPooledMemManager.FreedCount` (??)

25.5.12 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` (??), `TPooledMemManager.FreeCount` (??), `TPooledMemManager.AllocatedCount` (??)

Chapter 26

Reference for unit 'process'

26.1 Used units

Table 26.1: Used units by unit 'process'

Name	Page
Classes	??
Pipes	505
System	??
sysutils	??

26.2 Overview

The `Process` unit contains the code for the `TProcess` ([520](#)) 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.

26.3 Constants, types and variables

26.3.1 Types

`TProcessForkEvent` = procedure

`TProcessForkEvent` is the prototype for `TProcess.OnForkEvent` (??). 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 26.2: 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(??), 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 (517).

TProcessPriority = (ppHigh, ppIdle, ppNormal, ppRealTime)

Table 26.3: 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 26.4: 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 26.5: Enumeration values for type TStartupOption

Value	Explanation
suoUseCountChars	Use the console character width as specified in TProcess (520).
suoUseFillAttribute	Use the console fill attribute as specified in TProcess (520).
suoUsePosition	Use the window sizes as specified in TProcess (520).
suoUseShowWindow	Use the Show Window options specified in TShowWindowOption (517)
suoUseSize	Use the window sizes as specified in TProcess (520)

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 (518).

26.3.2 Variables

```
TryTerminals : Array of
```

TryTerminals is used under unix to test for available terminal programs in the DetectXTerm (519) function. If XTermProgram (518) 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 (519) is called.

26.4 Procedures and functions

26.4.1 CommandToList

Declaration: `procedure CommandToList (S: string; List: TStrings)`

Visibility: default

26.4.2 DetectXTerm

Synopsis: Detect the terminal program.

Declaration: `function DetectXTerm : string`

Visibility: default

Description: `DetectXTerm` checks if `XTermProgram` (518) is set. if so, it returns that. If `XTermProgram` is empty, the list specified in `TryTerminals` (518) is tested for existence. If none is found, then the `DESKTOP_SESSION` environment variable is examined:

kdekonsole is used if it is found.

gnome `gnome-terminal` is used if it is found

windowmaker `aterm` or `xterm` are used if found.

If after all this, no terminal is found, then a list of default programs is tested: `'x-terminal-emulator'`, `'xterm'`, `'aterm'`, `'wterm'`, `'rxvt'`

If a terminal program is found, then it is saved in `XTermProgram`, so the next call to `DetectXTerm` will re-use the value. If the search must be performed again, it is sufficient to set `XTermProgram` to the empty string.

See also: `XTermProgram` (518), `TryTerminals` (518), `TProcess.XTermProgram` (??)

26.4.3 RunCommand

Declaration: `function RunCommand(const exename: string; const commands: Array of ;
var outputstring: string) : Boolean`
`function RunCommand(const cmdline: string; var outputstring: string)
: Boolean`

Visibility: default

26.4.4 RunCommandIndir

Declaration: `function RunCommandIndir(const curdir: string; const exename: string;
const commands: Array of ;
var outputstring: string;
var exitstatus: Integer) : Integer`
`function RunCommandIndir(const curdir: string; const exename: string;
const commands: Array of ;
var outputstring: string) : Boolean`
`function RunCommandInDir(const curdir: string; const cmdline: string;
var outputstring: string) : Boolean`

Visibility: default

26.5 EProcess

26.5.1 Description

Exception raised when an error occurs in a TProcess routine.

See also: TProcess ([520](#))

26.6 TProcess

26.6.1 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 (??) property to the full pathname of the program that should be executed, and call Execute (??). To determine whether the process is still running (i.e. has not stopped executing), the Running (??) property can be checked.

More advanced techniques can be used with the Options (??) settings.

See also: Create (??), Execute (??), Running (??), CommandLin (??), Options (??)

26.6.2 Method overview

Page	Property	Description
522	CloseInput	Close the input stream of the process
523	CloseOutput	Close the output stream of the process
523	CloseStderr	Close the error stream of the process
521	Create	Create a new instance of the TProcess class.
522	Destroy	Destroy this instance of TProcess
522	Execute	Execute the program with the given options
523	Resume	Resume execution of a suspended process
523	Suspend	Suspend a running process
524	Terminate	Terminate a running process
524	WaitOnExit	Wait for the program to stop executing.

26.6.3 Property overview

Page	Property	Access	Description
528	Active	rw	Start or stop the process.
528	ApplicationName	rw	Name of the application to start (deprecated)
529	CommandLine	rw	Command-line to execute (deprecated)
530	ConsoleTitle	rw	Title of the console window
530	CurrentDirectory	rw	Working directory of the process.
531	Desktop	rw	Desktop on which to start the process.
531	Environment	rw	Environment variables for the new process
529	Executable	rw	Executable name. Supersedes <code>CommandLine</code> and <code>ApplicationName</code> .
527	ExitStatus	r	Exit status of the process.
536	FillAttribute	rw	Color attributes of the characters in the console window (Windows only)
524	Handle	r	Handle of the process
527	InheritHandles	rw	Should the created process inherit the open handles of the current process.
526	Input	r	Stream connected to standard input of the process.
528	OnForkEvent	rw	Event triggered after fork occurred on Linux
531	Options	rw	Options to be used when starting the process.
526	Output	r	Stream connected to standard output of the process.
529	Parameters	rw	Command-line arguments. Supersedes <code>CommandLine</code> .
528	PipeBufferSize	rw	
532	Priority	rw	Priority at which the process is running.
525	ProcessHandle	r	Alias for <code>Handle</code> (524)
525	ProcessID	r	ID of the process.
533	Running	r	Determines whether the process is still running.
533	ShowWindow	rw	Determines how the process main window is shown (Windows only)
533	StartupOptions	rw	Additional (Windows) startup options
527	Stderr	r	Stream connected to standard diagnostic output of the process.
525	ThreadHandle	r	Main process thread handle
526	ThreadID	r	ID of the main process thread
534	WindowColumns	rw	Number of columns in console window (Windows only)
534	WindowHeight	rw	Height of the process main window
535	WindowLeft	rw	X-coordinate of the initial window (Windows only)
524	WindowRect	rw	Positions for the main program window.
535	WindowRows	rw	Number of rows in console window (Windows only)
535	WindowTop	rw	Y-coordinate of the initial window (Windows only)
535	WindowWidth	rw	Height of the process main window (Windows only)
536	XTermProgram	rw	XTerm program to use (unix only)

26.6.4 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.

26.6.5 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 (??)`

26.6.6 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 (??)`, applying as much as of the specified options as supported on the current platform.

If the `poWaitOnExit` option is specified in `Options (??)`, 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 (??)` call can be used to wait for it to close, or the `Running (??)` call can be used to check whether it is still running.

The `TProcess.Terminate (??)` call can be used to terminate the program if it is still running, or the `Suspend (??)` call can be used to temporarily stop the program's execution.

The `ExitStatus (??)` function can be used to check the program's exit status, after it has stopped executing.

Errors: On error a `EProcess (520)` exception is raised.

See also: `TProcess.Running (??)`, `TProcess.WaitOnExit (??)`, `TProcess.Terminate (??)`, `TProcess.Suspend (??)`, `TProcess.Resume (??)`, `TProcess.ExitStatus (??)`

26.6.7 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 (??)`, `StdErr (??)`, `Output (??)`, `CloseOutput (??)`, `CloseStdErr (??)`

26.6.8 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 (??)`, `Input (??)`, `StdErr (??)`, `CloseInput (??)`, `CloseStdErr (??)`

26.6.9 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 (??)`, `Input (??)`, `StdErr (??)`, `CloseInput (??)`, `CloseStdErr (??)`

26.6.10 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 (??)`.

Errors: None.

See also: `TProcess.Suspend (??)`, `TProcess.Options (??)`, `TProcess.Execute (??)`, `TProcess.Terminate (??)`

26.6.11 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 (??)` call.

`Suspend` is fundamentally different from `TProcess.Terminate (??)` which actually stops the process.

Errors: On error, a nonzero result is returned.

See also: `TProcess.Options (??)`, `TProcess.Resume (??)`, `TProcess.Terminate (??)`, `TProcess.Execute (??)`

26.6.12 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` (??), `TProcess.Suspend` (??), `TProcess.Execute` (??), `TProcess.WaitOnExit` (??)

26.6.13 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 succesful, 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` (??), `TProcess.Terminate` (??), `TProcess.Running` (??)

26.6.14 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

26.6.15 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 (??)` has been called. It is not reset after the process stopped.

See also: `TProcess.ThreadHandle (??)`, `TProcess.ProcessID (??)`, `TProcess.ThreadID (??)`

26.6.16 TProcess.ProcessHandle

Synopsis: Alias for `Handle` ([524](#))

Declaration: `Property ProcessHandle : THandle`

Visibility: public

Access: Read

Description: `ProcessHandle` equals `Handle (??)` and is provided for completeness only.

See also: `TProcess.Handle (??)`, `TProcess.ThreadHandle (??)`, `TProcess.ProcessID (??)`, `TProcess.ThreadID (??)`

26.6.17 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 (??)` has been called. It is not reset after the process stopped.

See also: `TProcess.Handle (??)`, `TProcess.ProcessID (??)`, `TProcess.ThreadID (??)`

26.6.18 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 (??)` has been called. It is not reset after the process stopped.

See also: `TProcess.Handle (??)`, `TProcess.ThreadHandle (??)`, `TProcess.ThreadID (??)`

26.6.19 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 (??)` has been called. It is not reset after the process stopped.

See also: `TProcess.ProcessID (??)`, `TProcess.Handle (??)`, `TProcess.ThreadHandle (??)`

26.6.20 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 (??)`.

Note that writing to the stream may cause the calling process to be suspended when the created process is not reading from it's input, or to cause errors when the process has terminated.

See also: `TProcess.OutPut (??)`, `TProcess.StdErr (??)`, `TProcess.Options (??)`, `TProcessOption (517)`

26.6.21 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 (??)`.

The `Output` stream also contains any data written to standard diagnostic output (`stderr`) when the `poStdErrToOutPut` flag is used in `Options (??)`.

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 (??)`, `TProcess.StdErr (??)`, `TProcess.Options (??)`, `TProcessOption (517)`

26.6.22 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` (??).

The Output stream equals the Output (??) when the `poStdErrToOutPut` flag is used in `Options` (??).

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` (??), `TProcess.Output` (??), `TProcess.Options` (??), `TProcessOption` (517)

26.6.23 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` (??), `TProcess.Terminate` (??)

26.6.24 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` (??), `TProcess.Output` (??), `TProcess.StdErr` (??)

26.6.25 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` (??), `Input` (??), `StdErr` (??), `CloseInput` (??), `CloseStdErr` (??), `TProcessForkEvent` ([516](#))

26.6.26 TProcess.PipeBufferSize

Declaration: `Property PipeBufferSize : Cardinal`

Visibility: published

Access: Read,Write

26.6.27 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` (??), `TProcess.Terminate` (??)

26.6.28 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` (??). 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` (??) instead, and leave `ApplicationName` empty.

See also: `TProcess.CommandLine` (??), `TProcess.Executable` (??), `TProcess.Parameters` (??)

26.6.29 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` (??) and `TProcess.Parameters` (??), 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` (??), `TProcess.Executable` (??), `TProcess.Parameters` (??)

26.6.30 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` (??).

`Executable` supersedes the `TProcess.CommandLine` (??) and `TProcess.ApplicationName` (??) properties, which have been deprecated. However, if either of `CommandLine` or `ApplicationName` is specified, they will be used instead of `Executable`.

See also: `CommandLine` (??), `ApplicationName` (??), `Parameters` (??)

26.6.31 TProcess.Parameters

Synopsis: Command-line arguments. Supersedes `CommandLine`.

Declaration: `Property Parameters : TStrings`

Visibility: published

Access: Read,Write

Description: `Parameters` contains the command-line arguments that should be passed to the program specified in `Executable` (??).

Commandline arguments should be specified one per item in `Parameters`: each item in `Parameters` will be passed as a separate command-line item. It is therefor not necessary to quote whitespace in the items. As a consequence, it is not allowed to specify multiple command-line parameters in 1 item in the stringlist. If a command needs 2 options `-t` and `-s`, the following is not correct:

```
With Parameters do
begin
  add(' -t -s' );
end;
```

Instead, the code should read:

```
With Parameters do
begin
  add(' -t' );
  Add(' -s' );
end;
```

Remark: Note that `Parameters` is ignored if either of `CommandLine` or `ApplicationName` is specified. It can only be used with `Executable`.

Remark: The idea of using `Parameters` is that they are passed unmodified to the operating system. On Windows, a single command-line string must be constructed, and each parameter is surrounded by double quote characters if it contains a space. The programmer must not quote parameters with spaces.

See also: `Executable` (??), `CommandLine` (??), `ApplicationName` (??)

26.6.32 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` (??), `TProcess.WindowRows` (??)

26.6.33 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` (??)

26.6.34 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` (??), `TProcess.Output` (??), `TProcess.StdErr` (??)

26.6.35 TProcess.Environment

Synopsis: Environment variables for the new process

Declaration: `Property Environment : TStringList`

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` (??)

26.6.36 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` (??) call is made.

Table 26.6:

Option	Meaning
<code>poRunSuspended</code>	Start the process in suspended state.
<code>poWaitOnExit</code>	Wait for the process to terminate before returning.
<code>poUsePipes</code>	Use pipes to redirect standard input and output.
<code>poStderrToOutPut</code>	Redirect standard error to the standard output stream.
<code>poNoConsole</code>	Do not allow access to the console window for the process (Win32 only)
<code>poNewConsole</code>	Start a new console window for the process (Win32 only)
<code>poDefaultErrorMode</code>	Use default error handling.
<code>poNewProcessGroup</code>	Start the process in a new process group (Win32 only)
<code>poDebugProcess</code>	Allow debugging of the process (Win32 only)
<code>poDebugOnlyThisProcess</code>	Do not follow processes started by this process (Win32 only)

See also: `TProcessOption` (517), `TProcessOptions` (517), `TProcess.Priority` (??), `TProcess.StartupOptions` (??)

26.6.37 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 26.7:

Priority	Meaning
<code>ppHigh</code>	The process runs at higher than normal priority.
<code>ppIdle</code>	The process only runs when the system is idle (i.e. has nothing else to do)
<code>ppNormal</code>	The process runs at normal priority.
<code>ppRealTime</code>	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 26.8:

Priority	Nice value
<code>ppHigh</code>	20
<code>ppIdle</code>	20
<code>ppNormal</code>	0
<code>ppRealTime</code>	-20

See also: `TProcessPriority` ([517](#))

26.6.38 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 26.9:

Priority	Meaning
<code>suoUseShowWindow</code>	Use the Show Window options specified in <code>ShowWindow</code> (??)
<code>suoUseSize</code>	Use the specified window sizes
<code>suoUsePosition</code>	Use the specified window sizes.
<code>suoUseCountChars</code>	Use the specified console character width.
<code>suoUseFillAttribute</code>	Use the console fill attribute specified in <code>FillAttribute</code> (??).

See also: `TProcess.ShowWindow` (??), `TProcess.WindowHeight` (??), `TProcess.WindowWidth` (??), `TProcess.WindowLeft` (??), `TProcess.WindowTop` (??), `TProcess.WindowColumns` (??), `TProcess.WindowRows` (??), `TProcess.FillAttribute` (??)

26.6.39 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` (??), `TProcess.Active` (??), `TProcess.ExitStatus` (??)

26.6.40 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 26.10:

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

26.6.41 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 (??)

See also: TProcess.WindowHeight (??), TProcess.WindowWidth (??), TProcess.WindowLeft (??), TProcess.WindowTop (??), TProcess.WindowRows (??), TProcess.FillAttribute (??), TProcess.StartupOptions (??)

26.6.42 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 (??)

See also: TProcess.WindowWidth (??), TProcess.WindowLeft (??), TProcess.WindowTop (??), TProcess.WindowColumns (??), TProcess.WindowRows (??), TProcess.FillAttribute (??), TProcess.StartupOptions (??)

26.6.43 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` (??)

See also: `TProcess.WindowHeight` (??), `TProcess.WindowWidth` (??), `TProcess.WindowTop` (??), `TProcess.WindowColumns` (??), `TProcess.WindowRows` (??), `TProcess.FillAttribute` (??), `TProcess.StartupOptions` (??)

26.6.44 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` (??)

See also: `TProcess.WindowHeight` (??), `TProcess.WindowWidth` (??), `TProcess.WindowLeft` (??), `TProcess.WindowTop` (??), `TProcess.WindowColumns` (??), `TProcess.FillAttribute` (??), `TProcess.StartupOptions` (??)

26.6.45 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` (??)

See also: `TProcess.WindowHeight` (??), `TProcess.WindowWidth` (??), `TProcess.WindowLeft` (??), `TProcess.WindowColumns` (??), `TProcess.WindowRows` (??), `TProcess.FillAttribute` (??), `TProcess.StartupOptions` (??)

26.6.46 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` (??)

See also: `TProcess.WindowHeight` (??), `TProcess.WindowLeft` (??), `TProcess.WindowTop` (??), `TProcess.WindowColumns` (??), `TProcess.WindowRows` (??), `TProcess.FillAttribute` (??), `TProcess.StartupOptions` (??)

26.6.47 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` (??), `TProcess.WindowWidth` (??), `TProcess.WindowLeft` (??), `TProcess.WindowTop` (??), `TProcess.WindowColumns` (??), `TProcess.WindowRows` (??), `TProcess.StartupOptions` (??)

26.6.48 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` (??).

If none is specified, `DetectXTerm` (519) is used to detect the terminal program to use. the list specified in `TryTerminals` is tried. If none is found, then the `DESKTOP_SESSION` environment variable is examined:

kdekonsole is used if it is found.

gnomegnome-terminal is used if it is found

windowmakeraterm or xterm are used if found.

If after all this, no terminal is found, then a list of default programs is tested: 'x-terminal-emulator', 'xterm', 'aterm', 'wterm', 'rxvt'

See also: `TProcess.Options` (??), `DetectXTerm` (519)

Chapter 27

Reference for unit 'rttiutils'

27.1 Used units

Table 27.1: Used units by unit 'rttiutils'

Name	Page
Classes	??
StrUtils	??
System	??
sysutils	??
typinfo	??

27.2 Overview

The `rttiutils` unit is a unit providing simplified access to the RTTI information from published properties using the `TPropInfoList` ([539](#)) 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` ([542](#)) class can be used for this. The implementation is based on the `apputils` unit from `RXLib` by *AO ROSNO* and *Master-Bank*

27.3 Constants, types and variables

27.3.1 Constants

```
sPropNameDelimiter : string = '_'
```

Separator used when constructing section/key names

27.3.2 Types

```
TEraseSectEvent = procedure(const ASection: string) of object
```

`TEraseSectEvent` is used by `TPropsStorage` (542) 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 ASecton: string;const Item: string;
                        const Default: string) : string of object
```

`TReadStrEvent` is used by `TPropsStorage` (542) to read strings from a storage mechanism, in a .ini file like fashion: The call should read the string in `ASecton` with key `Item`, and if it does not exist, `Default` should be returned.

```
TWriteStrEvent = procedure(const ASecton: string;const Item: string;
                        const Value: string) of object
```

`TWriteStrEvent` is used by `TPropsStorage` (542) to write strings to a storage mechanism, in a .ini file like fashion: The call should write the string `Value` in `ASecton` with key `Item`. The section and key should be created if they didn't exist yet.

27.3.3 Variables

```
FindGlobalComponentCallBack : TFindComponentEvent
```

`FindGlobalComponentCallBack` is called by `UpdateStoredList` (539) 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.

27.4 Procedures and functions

27.4.1 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` (539), `TPropsStorage.StoreObjectsProps` (??) or `TPropsStorage.LoadObjectsProps` (??).

See also: `ParseStoredItem` (539), `UpdateStoredList` (539), `TPropsStorage.StoreObjectsProps` (??), `TPropsStorage.LoadObjectsProps` (??)

27.4.2 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` (538). 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` (538), `UpdateStoredList` (539), `TPropsStorage.StoreObjectsProps` (??), `TPropsStorage.LoadObjectsProps` (??)

27.4.3 UpdateStoredList

Synopsis: Update a stringlist with object references

Declaration: `procedure UpdateStoredList (AComponent: TComponent; AStoredList: TStringList;
FromForm: Boolean)`

Visibility: default

Description: `UpdateStoredList` will parse the strings in `AStoredList` using `ParseStoredItem` (539) 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` (538) 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` (539), `TPropsStorage.StoreObjectsProps` (??), `TPropsStorage.LoadObjectsProps` (??), `FindGlobalComponentCallBack` (538)

27.5 TPropInfoList

27.5.1 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` (542)

See also: `TPropsStorage` (542)

27.5.2 Method overview

Page	Property	Description
540	Contains	Check whether a certain property is included
540	Create	Create a new instance of <code>TPropInfoList</code>
541	Delete	Delete property information from the list
540	Destroy	Remove the <code>TPropInfoList</code> instance from memory
541	Find	Retrieve property information based on name
541	Intersect	Intersect 2 property lists

27.5.3 Property overview

Page	Property	Access	Description
541	Count	r	Number of items in the list
542	Items	r	Indexed access to the property type pointers

27.5.4 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` (??)

27.5.5 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` (??)

27.5.6 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` (??), `Intersect` (??)

27.5.7 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 (??)`, `Contains (??)`

27.5.8 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 (??)` call.

Errors: No checking on the validity of `Index` is performed.

See also: `Intersect (??)`

27.5.9 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 (??)`, `Contains (??)`

27.5.10 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 (??)`

27.5.11 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` (??)

27.6 TPropsStorage

27.6.1 Description

`TPropsStorage` provides a mechanism to store properties from any class which has published properties (usually a `TPersistent` descendent) in a storage mechanism.

`TPropsStorage` does not handle the storage by itself, instead, the storage is handled through a series of callbacks to read and/or write strings. Conversion of property types to string is handled by `TPropsStorage` itself: all that needs to be done is set the 3 handlers. The storage mechanism is assumed to have the structure of an .ini file : sections with key/value pairs. The three callbacks should take this into account, but they do not need to create an actual .ini file.

See also: `TPropInfoList` ([539](#))

27.6.2 Method overview

Page	Property	Description
543	<code>LoadAnyProperty</code>	Load a property value
544	<code>LoadObjectsProps</code>	Load a list of component properties
543	<code>LoadProperties</code>	Load a list of properties
542	<code>StoreAnyProperty</code>	Store a property value
544	<code>StoreObjectsProps</code>	Store a list of component properties
543	<code>StoreProperties</code>	Store a list of properties

27.6.3 Property overview

Page	Property	Access	Description
545	<code>AObject</code>	rw	Object to load or store properties from
546	<code>OnEraseSection</code>	rw	Erase a section in storage
546	<code>OnReadString</code>	rw	Read a string value from storage
546	<code>OnWriteString</code>	rw	Write a string value to storage
545	<code>Prefix</code>	rw	Prefix to use in storage
545	<code>Section</code>	rw	Section name for storage

27.6.4 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 (??)` property of `TPropsStorage`.

Errors: If the property pointer is invalid or `AObject` is invalid, an exception will be raised.

See also: `AObject (??)`, `LoadAnyProperty (??)`, `LoadProperties (??)`, `StoreProperties (??)`

27.6.5 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 (??)` property of `TPropsStorage`.

Errors: If the property pointer is invalid or `AObject` is invalid, an exception will be raised.

See also: `AObject (??)`, `StoreAnyProperty (??)`, `LoadProperties (??)`, `StoreProperties (??)`

27.6.6 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 (??)` object.

Errors: If an invalid property name is specified, an exception will be raised.

See also: `AObject (??)`, `StoreAnyProperty (??)`, `LoadProperties (??)`, `LoadAnyProperty (??)`

27.6.7 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 (??)` object.

Errors: If an invalid property name is specified, an exception will be raised.

See also: `AObject (??)`, `StoreAnyProperty (??)`, `StoreProperties (??)`, `LoadAnyProperty (??)`

27.6.8 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` (539) 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 (??)` property.

Errors: If an invalid component is specified, an exception will be raised.

See also: `UpdateStoredList` (539), `StoreObjectsProps (??)`, `LoadProperties (??)`, `LoadAnyProperty (??)`

27.6.9 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` (539) 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 (??) property.

See also: UpdateStoredList (539), LoadObjectsProps (??), LoadProperties (??), LoadAnyProperty (??)

27.6.10 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 (??) or LoadObjectProps (??) will destroy any value that this property might have.

See also: LoadProperties (??), LoadAnyProperty (??), StoreProperties (??), StoreAnyProperty (??), StoreObjectProps (??), LoadObjectProps (??)

27.6.11 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 (??)

27.6.12 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 (??)

27.6.13 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` (??), `OnEraseSection` (??), `TReadStrEvent` ([538](#))

27.6.14 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` (??), `OnEraseSection` (??), `TWriteStrEvent` ([538](#))

27.6.15 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` (??), `OnWriteString` (??), `TEraseSectEvent` ([537](#))

Chapter 28

Reference for unit 'simpleipc'

28.1 Used units

Table 28.1: Used units by unit 'simpleipc'

Name	Page
Classes	??
System	??
sysutils	??

28.2 Overview

The SimpleIPC unit provides classes to implement a simple, one-way IPC mechanism using string messages. It provides a TSimpleIPCServer (558) component for the server, and a TSimpleIPCClient (555) 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.

28.3 Constants, types and variables

28.3.1 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

28.3.2 Constants

```
MsgVersion = 1
```

Current version of the messaging protocol

```
mtString = 1
```

String message type

```
mtUnknown = 0
```

Unknown message type

28.3.3 Types

```
TIPCClientCommClass = Class of TIPCClientComm
```

`TIPCClientCommClass` is used by `TSimpleIPCClient` (555) to decide which kind of communication channel to set up.

```
TIPCServerCommClass = Class of TIPCServerComm
```

`TIPCServerCommClass` is used by `TSimpleIPCServer` (558) 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.

28.3.4 Variables

```
DefaultIPCClientClass : TIPCCClientCommClass = Nil
```

`DefaultIPCClientClass` is filled with a class pointer indicating which kind of communication protocol class should be instantiated by the `TSimpleIPCCClient` (555) 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` (558) 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.

28.4 EIPCErrors

28.4.1 Description

`EIPCErrors` is the exception used by the various classes in the `SimpleIPC` unit to report errors.

28.5 TIPCCClientComm

28.5.1 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` (555) 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` (551) component. The descendent components must always be implemented in pairs.

See also: `TSimpleIPCCClient` (555), `TIPCServerComm` (551), `TSimpleIPCServer` (558)

28.5.2 Method overview

Page	Property	Description
550	<code>Connect</code>	Connect to the server
550	<code>Create</code>	Create a new instance of the <code>TIPCCClientComm</code>
550	<code>Disconnect</code>	Disconnect from the server
551	<code>SendMessage</code>	Send a message
551	<code>ServerRunning</code>	Check if the server is running.

28.5.3 Property overview

Page	Property	Access	Description
551	<code>Owner</code>	<code>r</code>	<code>TSimpleIPCCClient</code> instance for which communication must be handled.

28.5.4 TIPCCliantComm.Create

Synopsis: Create a new instance of the `TIPCCliantComm`

Declaration: `constructor Create(AOwner: TSimpleIPCCliant); Virtual`

Visibility: `public`

Description: `Create` instantiates a new instance of the `TIPCCliantComm` class, and stores the `AOwner` reference to the `TSimpleIPCCliant` (555) instance for which it will handle communication. It can be retrieved later using the `Owner` (??) property.

See also: `Owner` (??), `TSimpleIPCCliant` (555)

28.5.5 TIPCCliantComm.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` (??) and `ServerInstance` (??) properties of the owning `TSimpleIPCCliant` (555) instance.

`Connect` is called by the `TSimpleIPCCliant.Connect` (??) call or when the `Active` (??) 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: `TSimpleIPCCliant.Connect` (??), `Active` (??), `Disconnect` (??)

28.5.6 TIPCCliantComm.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 `TSimpleIPCCliant.Disconnect` (??) call or when the `Active` (??) 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: `TSimpleIPCCliant.Disconnect` (??), `Active` (??), `Connect` (??)

28.5.7 TIPCCliantComm.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 `TSimpleIPCCliant` (555) component.

See also: `TSimpleIPCCliant.InstanceID` (??), `TSimpleIPCCliant.ServerID` (??)

28.5.8 TIPCCliantComm.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.

28.5.9 TIPCCliantComm.Owner

Synopsis: `TSimpleIPCCliant` instance for which communication must be handled.

Declaration: `Property Owner : TSimpleIPCCliant`

Visibility: `public`

Access: `Read`

Description: `Owner` is the `TSimpleIPCCliant` (555) instance for which the communication must be handled. It cannot be changed, and must be specified when the `TIPCCliantComm` instance is created.

See also: `TSimpleIPCCliant` (555), `TIPCCliantComm.Create` (??)

28.6 TIPCTServerComm

28.6.1 Description

`TIPCTServerComm` 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 `TSimpleIPCTServer` (558) class does not implement the messaging protocol by itself. Instead, it creates an instance of a (platform dependent) descendent of `TIPCTServerComm` which handles the internals of the communication protocol.

The client side of the messaging protocol is handled by the `TIPCCliantComm` (549) component. The descendent components must always be implemented in pairs.

See also: `TSimpleIPCTServer` (558), `TIPCCliantComm` (549)

28.6.2 Method overview

Page	Property	Description
552	Create	Create a new instance of the communication handler
553	PeekMessage	See if a message is available.
553	ReadMessage	Read message from the channel.
552	StartServer	Start the server-side of the communication channel
552	StopServer	Stop the server side of the communication channel.

28.6.3 Property overview

Page	Property	Access	Description
554	InstanceID	r	Unique identifier for the communication channel.
553	Owner	r	TSimpleIPCServer instance for which to handle transport

28.6.4 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` (??) property.

See also: `Owner` (??)

28.6.5 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` (??) property of the `TSimpleIPCServer` ([558](#)) instance is set to `True`.

Errors: In case of an error, an `EIPCError` ([549](#)) exception is raised.

See also: `TSimpleIPCServer` ([558](#)), `TSimpleIPC.Active` (??)

28.6.6 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 (??)` property of the `TSimpleIPCServer (558)` instance is set to `False`.

Errors: In case of an error, an `EIPCError (549)` exception is raised.

See also: `TSimpleIPCServer (558)`, `TSimpleIPC.Active (??)`

28.6.7 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 (??)` call.

See also: `ReadMessage (??)`

28.6.8 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 (??)` to see if a message is available.

See also: `TSimpleIPCServer (558)`

28.6.9 TIPCServerComm.Owner

Synopsis: `TSimpleIPCServer` instance for which to handle transport

Declaration: `Property Owner : TSimpleIPCServer`

Visibility: `public`

Access: `Read`

Description: `Owner` refers to the `TSimpleIPCServer (558)` instance for which this instance of `TSimpleIPCServer` handles the transport. It is specified when the `TIPCServerComm` is created.

See also: `TSimpleIPCServer (558)`

28.6.10 TIPServerComm.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.

28.7 TSimpleIPC

28.7.1 Description

`TSimpleIPC` is the common ancestor for the `TSimpleIPCServer` (558) and `TSimpleIPCClient` (555) classes. It implements some common properties between client and server.

See also: `TSimpleIPCServer` (558), `TSimpleIPCClient` (555)

28.7.2 Property overview

Page	Property	Access	Description
554	<code>Active</code>	<code>rw</code>	Communication channel active
554	<code>ServerID</code>	<code>rw</code>	Unique server identification

28.7.3 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` (??) identification.

See also: `ServerID` (??)

28.7.4 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` (??)

28.8 TSimpleIPCClient

28.8.1 Description

`TSimpleIPCClient` is the client side of the simple IPC communication protocol. The client program should create a `TSimpleIPCClient` instance, set its `ServerID` (??) property to the unique name for the server it wants to send messages to, and then set the `Active` (??) property to `True` (or call `Connect` (??)).

After the connection with the server was established, messages can be sent to the server with the `SendMessage` (??) or `SendStringMessage` (??) calls.

See also: `TSimpleIPCServer` (558), `TSimpleIPC` (554), `TIPCClientComm` (549)

28.8.2 Method overview

Page	Property	Description
556	<code>Connect</code>	Connect to the server
555	<code>Create</code>	Create a new instance of <code>TSimpleIPCClient</code>
555	<code>Destroy</code>	Remove the <code>TSimpleIPCClient</code> instance from memory
556	<code>Disconnect</code>	Disconnect from the server
557	<code>SendMessage</code>	Send a message to the server
557	<code>SendStringMessage</code>	Send a string message to the server
557	<code>SendStringMessageFmt</code>	Send a formatted string message
556	<code>ServerRunning</code>	Check if the server is running.

28.8.3 Property overview

Page	Property	Access	Description
557	<code>ServerInstance</code>	rw	Server instance identification

28.8.4 TSimpleIPCClient.Create

Synopsis: Create a new instance of `TSimpleIPCClient`

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

Visibility: `public`

Description: `Create` instantiates a new instance of the `TSimpleIPCClient` class. It initializes the data structures needed to handle the client side of the communication.

See also: `Destroy` (??)

28.8.5 TSimpleIPCClient.Destroy

Synopsis: Remove the `TSimpleIPCClient` instance from memory

Declaration: `destructor Destroy; Override`

Visibility: `public`

Description: `Destroy` disconnects the client from the server if need be, and cleans up the internal data structures maintained by `TSimpleIPCClient` and then calls the inherited `Destroy`, which will remove the instance from memory.

Never call `Destroy` directly, use the `Free` method instead or the `FreeAndNil` procedure in `SysUtils`.

See also: `Create` (??)

28.8.6 TSimpleIPClient.Connect

Synopsis: Connect to the server

Declaration: `procedure Connect`

Visibility: `public`

Description: `Connect` connects to the server indicated in the `ServerID` (??) and `InstanceID` (??) properties. `Connect` is called automatically if the `Active` (??) property is set to `True`.

After a successful call to `Connect`, messages can be sent to the server using `SendMessage` (??) or `SendStringMessage` (??).

Calling `Connect` if the connection is already open has no effect.

Errors: If creating the connection fails, an `EIPCErr` (549) exception may be raised.

See also: `ServerID` (??), `InstanceID` (??), `Active` (??), `SendMessage` (??), `SendStringMessage` (??), `Disconnect` (??)

28.8.7 TSimpleIPClient.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` (??). `Disconnect` is called automatically if the `Active` (??) 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 `EIPCErr` (549) exception may be raised.

See also: `Active` (??), `Connect` (??)

28.8.8 TSimpleIPClient.ServerRunning

Synopsis: Check if the server is running.

Declaration: `function ServerRunning : Boolean`

Visibility: `public`

Description: `ServerRunning` verifies if the server indicated in the `ServerID` (??) and `InstanceID` (??) 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` (??)

28.8.9 TSimpleIPCClient.SendMessage

Synopsis: Send a message to the server

Declaration: `procedure SendMessage(MsgType: TMessageType; Stream: TStream)`

Visibility: public

Description: `SendMessage` sends a message of type `MsgType` and data from `stream` to the server. The client must be connected for this call to work.

Errors: In case an error occurs, or there is no connection to the server, an `EIPCErr` (549) exception is raised.

See also: `Connect` (??), `SendStringMessage` (??)

28.8.10 TSimpleIPCClient.SendStringMessage

Synopsis: Send a string message to the server

Declaration: `procedure SendStringMessage(const Msg: string)`
`procedure SendStringMessage(MsgType: TMessageType; const Msg: string)`

Visibility: public

Description: `SendStringMessage` sends a string message with type `MsgType` and data `Msg` to the server. This is a convenience function: a small wrapper around the `SendMessage` (??) method

Errors: Same as for `SendMessage`.

See also: `SendMessage` (??), `Connect` (??), `SendStringMessageFmt` (??)

28.8.11 TSimpleIPCClient.SendStringMessageFmt

Synopsis: Send a formatted string message

Declaration: `procedure SendStringMessageFmt(const Msg: string; Args: Array of const)`
`procedure SendStringMessageFmt(MsgType: TMessageType; const Msg: string; Args: Array of const)`

Visibility: public

Description: `SendStringMessageFmt` sends a string message with type `MsgType` and message formatted from `Msg` and `Args` to the server. This is a convenience function: a small wrapper around the `SendStringMessage` (??) method

Errors: Same as for `SendMessage`.

See also: `SendMessage` (??), `Connect` (??), `SendStringMessage` (??)

28.8.12 TSimpleIPCClient.ServerInstance

Synopsis: Server instance identification

Declaration: `Property ServerInstance : string`

Visibility: public

Access: Read, Write

Description: `ServerInstance` should be used in case a particular instance of the server identified with `ServerID` should be contacted. This must be used if the server has its `Global (??)` property set to `False`, and should match the server's `InstanceID (??)` property.

See also: `ServerID (??)`, `Global (??)`, `InstanceID (??)`

28.9 TSimpleIPCServer

28.9.1 Description

`TSimpleIPCServer` is the server side of the simple IPC communication protocol. The server program should create a `TSimpleIPCServer` instance, set its `ServerID (??)` property to a unique name for the system, and then set the `Active (??)` property to `True` (or call `StartServer (??)`).

After the server was started, it can check for availability of messages with the `PeekMessage (??)` call, and read the message with `ReadMessage (??)`.

See also: `TSimpleIPCClient (555)`, `TSimpleIPC (554)`, `TIPCServerComm (551)`

28.9.2 Method overview

Page	Property	Description
558	Create	Create a new instance of <code>TSimpleIPCServer</code>
559	Destroy	Remove the <code>TSimpleIPCServer</code> instance from memory
560	GetMessageData	Read the data of the last message in a stream
559	PeekMessage	Check if a client message is available.
559	StartServer	Start the server
559	StopServer	Stop the server

28.9.3 Property overview

Page	Property	Access	Description
561	Global	rw	Is the server reachable to all users or not
561	InstanceID	r	Instance ID
561	MsgData	r	Last message data
560	MsgType	r	Last message type
561	OnMessage	rw	Event triggered when a message arrives
560	StringMessage	r	Last message as a string.

28.9.4 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 (??)`

28.9.5 TSimpleIPCTServer.Destroy

Synopsis: Remove the TSimpleIPCTServer instance from memory

Declaration: `destructor Destroy; Override`

Visibility: `public`

Description: `Destroy` stops the server, cleans up the internal data structures maintained by TSimpleIPCTServer 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 (??)`

28.9.6 TSimpleIPCTServer.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 `TIPCTServerComm` (551) descendent) and activates the communication channel.

After this method was called, clients can connect and send messages.

Prior to calling this method, the `ServerID (??)` property must be set.

Errors: If an error occurs a `EIPCTError` (549) exception may be raised.

See also: `TIPCTServerComm` (551), `Active (??)`, `ServerID (??)`, `StopServer (??)`

28.9.7 TSimpleIPCTServer.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 `TIPCTServerComm` (551) descendent).

See also: `TIPCTServerComm` (551), `Active (??)`, `ServerID (??)`, `StartServer (??)`

28.9.8 TSimpleIPCTServer.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 (??)`.

See also: `ReadMessage (??)`

28.9.9 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 `TSimpleIPCServer.MsgData (??)` 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 (??)`. It will also not clear the data buffer.

See also: `StringMessage (??)`, `MsgData (??)`, `MsgType (??)`

28.9.10 TSimpleIPCServer.StringMessage

Synopsis: Last message as a string.

Declaration: `Property StringMessage : string`

Visibility: `public`

Access: `Read`

Description: `StringMessage` is the content of the last message as a string.

This property will contain valid data only after a succesful call to `ReadMessage (??)`.

See also: `GetMessageData (??)`

28.9.11 TSimpleIPCServer.MsgType

Synopsis: Last message type

Declaration: `Property MsgType : TMessageType`

Visibility: `public`

Access: `Read`

Description: `MsgType` contains the message type of the last message.

This property will contain valid data only after a succesful call to `ReadMessage (??)`.

See also: `ReadMessage (??)`

28.9.12 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 (??)` is better suited to read the data.

This property will contain valid data only after a succesful call to `ReadMessage (??)`.

See also: `StringMessage (??)`, `ReadMessage (??)`

28.9.13 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 (??)` property to form the unique server endpoint which a client should use.

See also: `ServerID (??)`, `Global (??)`

28.9.14 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 (??)`, `InstanceID (??)`

28.9.15 TSimpleIPCServer.OnMessage

Synopsis: Event triggered when a message arrives

Declaration: `Property OnMessage : TNotifyEvent`

Visibility: published

Access: Read,Write

Description: `OnMessage` is called by `ReadMessage (??)` when a message has been read. The actual message data can be retrieved with one of the `StringMessage (??)`, `MsgData (??)` or `MsgType (??)` properties.

See also: `StringMessage (??)`, `MsgData (??)`, `MsgType (??)`

Chapter 29

Reference for unit 'streamcoll'

29.1 Used units

Table 29.1: Used units by unit 'streamcoll'

Name	Page
Classes	??
System	??
sysutils	??

29.2 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` (565) 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.

29.3 Procedures and functions

29.3.1 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` (564) and returns the read value. The value cannot be read and written across systems that have different endian values.

See also: [ColReadDateTime \(563\)](#), [ColWriteBoolean \(564\)](#), [ColReadString \(564\)](#), [ColReadInteger \(563\)](#), [ColReadFloat \(563\)](#), [ColReadCurrency \(563\)](#)

29.3.2 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 (564)` and returns the read value. The value cannot be read and written across systems that have different endian values.

See also: [ColReadDateTime \(563\)](#), [ColReadBoolean \(562\)](#), [ColReadString \(564\)](#), [ColReadInteger \(563\)](#), [ColReadFloat \(563\)](#), [ColWriteCurrency \(564\)](#)

29.3.3 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 (564)` and returns the read value. The value cannot be read and written across systems that have different endian values.

See also: [ColWriteDateTime \(564\)](#), [ColReadBoolean \(562\)](#), [ColReadString \(564\)](#), [ColReadInteger \(563\)](#), [ColReadFloat \(563\)](#), [ColReadCurrency \(563\)](#)

29.3.4 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 (565)` and returns the read value. The value cannot be read and written across systems that have different endian values.

See also: [ColReadDateTime \(563\)](#), [ColReadBoolean \(562\)](#), [ColReadString \(564\)](#), [ColReadInteger \(563\)](#), [ColWriteFloat \(565\)](#), [ColReadCurrency \(563\)](#)

29.3.5 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` (565) and returns the read value. The value cannot be read and written across systems that have different endian values.

See also: `ColReadDateTime` (563), `ColReadBoolean` (562), `ColReadString` (564), `ColWriteInteger` (565), `ColReadFloat` (563), `ColReadCurrency` (563)

29.3.6 ColReadString

Synopsis: Read a string from a stream

Declaration: `function ColReadString(S: TStream) : string`

Visibility: default

Description: `ColReadStream` reads a string value from the stream `S` as it was written by `ColWriteString` (565) and returns the read value. The value cannot be read and written across systems that have different endian values.

See also: `ColReadDateTime` (563), `ColReadBoolean` (562), `ColWriteString` (565), `ColReadInteger` (563), `ColReadFloat` (563), `ColReadCurrency` (563)

29.3.7 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` (562), `ColWriteString` (565), `ColWriteInteger` (565), `ColWriteCurrency` (564), `ColWriteDateTime` (564), `ColWriteFloat` (565)

29.3.8 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` (564), `ColWriteString` (565), `ColWriteInteger` (565), `ColWriteDateTime` (564), `ColWriteFloat` (565), `ColReadCurrency` (563)

29.3.9 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` (563), `ColWriteBoolean` (564), `ColWriteString` (565), `ColWriteInteger` (565), `ColWriteFloat` (565), `ColWriteCurrency` (564)

29.3.10 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` (564), `ColWriteBoolean` (564), `ColWriteString` (565), `ColWriteInteger` (565), `ColReadFloat` (563), `ColWriteCurrency` (564)

29.3.11 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` (564), `ColWriteString` (565), `ColReadInteger` (563), `ColWriteCurrency` (564), `ColWriteDateTime` (564)

29.3.12 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` (564), `ColReadStream` (564), `ColWriteInteger` (565), `ColWriteCurrency` (564), `ColWriteDateTime` (564), `ColWriteFloat` (565)

29.4 EStreamColl

29.4.1 Description

Exception raised when an error occurs when streaming the collection.

29.5 TStreamCollection

29.5.1 Description

`TStreamCollection` is a `TCollection` (??) descendent which implements 2 calls `LoadFromStream` (??) and `SaveToStream` (??) which load and save the contents of the collection to a stream.

The collection items must be descendents of the `TStreamCollectionItem` (567) class for the streaming to work correctly.

Note that the stream must be used to load collections of the same type.

See also: `TStreamCollectionItem` (567)

29.5.2 Method overview

Page	Property	Description
566	LoadFromStream	Load the collection from a stream
566	SaveToStream	Load the collection from the stream.

29.5.3 Property overview

Page	Property	Access	Description
566	Streaming	r	Indicates whether the collection is currently being written to stream

29.5.4 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` (??). 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` (??)

29.5.5 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` (??). 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` (??)

29.5.6 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` (??), `TStreamCollection.SaveToStream` (??)

29.6 TStreamCollectionItem

29.6.1 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 ([565](#)) to save or load the item from the stream.

See also: TStreamCollection ([565](#))

Chapter 30

Reference for unit 'streamex'

30.1 Used units

Table 30.1: Used units by unit 'streamex'

Name	Page
Classes	??
System	??

30.2 Overview

streamex implements some extensions to be used together with streams from the classes unit.

30.3 TBidirBinaryObjectReader

30.3.1 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 ([569](#)), TDelphiReader ([569](#))

30.3.2 Property overview

Page	Property	Access	Description
568	Position	rw	Position in the stream

30.3.3 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` (569) class.

See also: `TDelphiReader` (569)

30.4 TBidirBinaryObjectWriter

30.4.1 Description

`TBidirBinaryObjectReader` is a class descendent from `TBinaryObjectWriter` (??), which implements the necessary support for BiDi data.

See also: `TBidirBinaryObjectWriter` (569), `TDelphiWriter` (571)

30.4.2 Property overview

Page	Property	Access	Description
569	<code>Position</code>	rw	Position in the stream

30.4.3 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` (571) class.

See also: `TDelphiWriter` (571)

30.5 TDelphiReader

30.5.1 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` (568) driver class.

See also: `TDelphiWriter` (571), `TBidirBinaryObjectReader` (568)

30.5.2 Method overview

Page	Property	Description
570	<code>GetDriver</code>	Return the driver class as a <code>TBidirBinaryObjectReader</code> (568) class
570	<code>Read</code>	Read data from stream
570	<code>ReadStr</code>	Overrides the standard <code>ReadStr</code> method

30.5.3 Property overview

Page	Property	Access	Description
570	Position	rw	Position in the stream

30.5.4 TDelphiReader.GetDriver

Synopsis: Return the driver class as a `TBidirBinaryObjectReader` ([568](#)) class

Declaration: `function GetDriver : TBidirBinaryObjectReader`

Visibility: `public`

Description: `GetDriver` simply returns the used driver and typecasts it as `TBidirBinaryObjectReader` ([568](#)) class.

See also: `TBidirBinaryObjectReader` ([568](#))

30.5.5 TDelphiReader.ReadStr

Synopsis: Overrides the standard `ReadStr` method

Declaration: `function ReadStr : string`

Visibility: `public`

Description: `ReadStr` makes sure the `TBidirBinaryObjectReader` ([568](#)) methods are used, to store additional information about the stream position when reading the strings.

See also: `TBidirBinaryObjectReader` ([568](#))

30.5.6 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` ([568](#)) class when reading.

See also: `TBidirBinaryObjectReader` ([568](#)), `TDelphiReader.Position` (??)

30.5.7 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` (??)

30.6 TDelphiWriter

30.6.1 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` (569) driver class.

See also: `TDelphiReader` (569), `TBidirBinaryObjectWriter` (569)

30.6.2 Method overview

Page	Property	Description
571	<code>FlushBuffer</code>	Flushes the stream buffer
571	<code>GetDriver</code>	Return the driver class as a <code>TBidirBinaryObjectWriter</code> (569) class
571	<code>Write</code>	Write raw data to the stream
572	<code>WriteStr</code>	Write a string to the stream
572	<code>WriteValue</code>	Write value type

30.6.3 Property overview

Page	Property	Access	Description
572	<code>Position</code>	rw	Position in the stream

30.6.4 TDelphiWriter.GetDriver

Synopsis: Return the driver class as a `TBidirBinaryObjectWriter` (569) class

Declaration: `function GetDriver : TBidirBinaryObjectWriter`

Visibility: public

Description: `GetDriver` simply returns the used driver and typecasts it as `TBidirBinaryObjectWriter` (569) class.

See also: `TBidirBinaryObjectWriter` (569)

30.6.5 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.

30.6.6 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.

30.6.7 TDelphiWriter.WriteString

Synopsis: Write a string to the stream

Declaration: `procedure WriteStr(const Value: string)`

Visibility: public

Description: `WriteStr` writes a string to the stream, forcing the use of the `TBidirBinaryObjectWriter` (569) class methods, which update the position of the stream.

See also: `TBidirBinaryObjectWriter` (569)

30.6.8 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` (569) methods, which update the position of the stream.

See also: `TBidirBinaryObjectWriter` (569)

30.6.9 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` (569) instance used when streaming.

See also: `TBidirBinaryObjectWriter` (569)

Chapter 31

Reference for unit 'StreamIO'

31.1 Used units

Table 31.1: Used units by unit 'StreamIO'

Name	Page
Classes	??
System	??
sysutils	??

31.2 Overview

The `StreamIO` unit implements a call to reroute the input or output of a text file to a descendent of `TStream` (??).

This allows to use the standard pascal `Read` (??) and `Write` (??) functions (with all their possibilities), on streams.

31.3 Procedures and functions

31.3.1 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` ([574](#))

31.3.2 GetStream

Synopsis: Return the stream, associated with a file.

Declaration: `function GetStream(var F: TTextRec) : TStream`

Visibility: default

Description: `GetStream` returns the instance of the stream that was associated with the file `F` using `AssignStream` ([573](#)).

Errors: An invalid class reference will be returned if the file was not associated with a stream.

See also: `AssignStream` ([573](#)), `TStream` (??)

Chapter 32

Reference for unit 'syncobjs'

32.1 Used units

Table 32.1: Used units by unit 'syncobjs'

Name	Page
System	??
sysutils	??

32.2 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` ([576](#)) class is a wrapper around low-level critical section routines (semaphores or mutexes). The `TEventObject` ([578](#)) class can be used to send messages between threads (also known as conditional variables in Posix threads).

32.3 Constants, types and variables

32.3.1 Constants

`INFINITE = (-1)`

Constant denoting an infinite timeout.

32.3.2 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` ([578](#)) class.

`TEventHandle = Pointer`

`TEventHandle` is an opaque type and should not be used in user code.

`TWaitResult = (wrSignaled, wrTimeout, wrAbandoned, wrError)`

Table 32.2: Enumeration values for type `TWaitResult`

Value	Explanation
<code>wrAbandoned</code>	Wait operation was abandoned.
<code>wrError</code>	An error occurred during the wait operation.
<code>wrSignaled</code>	Event was signaled (triggered)
<code>wrTimeout</code>	Time-out period expired

`TWaitResult` is used to report the result of a wait operation.

32.4 TCriticalSection

32.4.1 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 (??)`, `Release (??)`

32.4.2 Method overview

Page	Property	Description
577	<code>Acquire</code>	Enter the critical section
578	<code>Create</code>	Create a new critical section.
578	<code>Destroy</code>	Destroy the criticalsection instance
577	<code>Enter</code>	Alias for <code>Acquire</code>
578	<code>Leave</code>	Alias for <code>Release</code>
577	<code>Release</code>	Leave the critical section
577	<code>TryEnter</code>	Try and obtain the critical section

32.4.3 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` (??)

32.4.4 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` (??)

32.4.5 TCriticalSection.Enter

Synopsis: Alias for `Acquire`

Declaration: `procedure Enter`

Visibility: `public`

Description: `Enter` just calls `Acquire` (??).

See also: `Leave` (??), `Acquire` (??)

32.4.6 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` (??)

32.4.7 TCriticalSection.Leave

Synopsis: Alias for Release

Declaration: `procedure Leave`

Visibility: `public`

Description: `Leave` just calls `Release` (??)

See also: `Release` (??), `Enter` (??)

32.4.8 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` (??)

32.4.9 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` (??), `Acquire` (??)

32.5 TEventObject

32.5.1 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` ([576](#))

32.5.2 Method overview

Page	Property	Description
579	Create	Create a new event object
579	destroy	Clean up the event and release from memory
579	ResetEvent	Reset the event
580	SetEvent	Set the event
580	WaitFor	Wait for the event to be set.

32.5.3 Property overview

Page	Property	Access	Description
580	ManualReset	r	Should the event be reset manually

32.5.4 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 immediatly after the first thread waiting for it is notified). `InitialState` determines whether the event is initially set or not.

See also: `ManualReset` (??), `ResetEvent` (??)

32.5.5 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` (??)

32.5.6 TEventObject.ResetEvent

Synopsis: Reset the event

Declaration: procedure `ResetEvent`

Visibility: public

Description: `ResetEvent` turns off the event. Any `WaitFor` (??) operation will suspend the calling thread.

See also: `SetEvent` (??), `WaitFor` (??)

32.5.7 TEventObject.SetEvent

Synopsis: Set the event

Declaration: `procedure SetEvent`

Visibility: `public`

Description: `SetEvent` sets the event. If the `ManualReset (??)` is `True` any thread that was waiting for the event to be set (using `WaitFor (??)`) will resume it's operation. After the event was set, any thread that executes `WaitFor` will return at once. If `ManualReset` is `False`, only one thread will be notified that the event was set, and the event will be immediatly reset after that.

See also: `WaitFor (??)`, `ManualReset (??)`

32.5.8 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 (??)`) 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 (??)`

32.5.9 TEventObject.ManualReset

Synopsis: Should the event be reset manually

Declaration: `Property ManualReset : Boolean`

Visibility: `public`

Access: `Read`

Description: Should the event be reset manually

32.6 THandleObject

32.6.1 Description

`THandleObject` is a parent class for synchronization classes that need to store an operating system handle. It introduces a property `Handle (??)` 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 (??)`

32.6.2 Method overview

Page	Property	Description
581	destroy	Free the instance

32.6.3 Property overview

Page	Property	Access	Description
581	Handle	r	Handle for this object
581	LastError	r	Last operating system error

32.6.4 THandleObject.destroy

Synopsis: Free the instance

Declaration: `destructor destroy; Override`

Visibility: `public`

Description: `Destroy` does nothing in the Free Pascal implementation of `THandleObject`.

32.6.5 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, descendent classes should set the handle by accessing it's protected field `FHandle` directly.

32.6.6 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` (??).

See also: `Handle` (??)

32.7 TSimpleEvent

32.7.1 Description

`TSimpleEvent` is a simple descendent of the `TEventObject` ([578](#)) class. It creates an event with no name, which must be reset manually, and which is initially not set.

See also: `TEventObject` ([578](#)), `TSimpleEvent.Create` (??)

32.7.2 Method overview

Page	Property	Description
582	Create	Creates a new <code>TSimpleEvent</code> instance

32.7.3 `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 (??)` with `Nil` for the security attributes, an empty name, `AManualReset` set to `True`, and `InitialState` to `False`.

See also: `TEventObject.Create (??)`

32.8 `TSynchroObject`

32.8.1 Description

`TSynchroObject` is an abstract synchronization resource object. It implements 2 virtual methods `Acquire (??)` which can be used to acquire the resource, and `Release (??)` to release the resource.

See also: `Acquire (??)`, `Release (??)`

32.8.2 Method overview

Page	Property	Description
582	Acquire	Acquire synchronization resource
582	Release	Release previously acquired synchronization resource

32.8.3 `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 (??)`

32.8.4 `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 (??)` call.

See also: `Acquire (??)`

Chapter 33

Reference for unit 'URIParser'

33.1 Overview

The `URIParser` unit contains a basic type (`TURI` ([584](#))) and some routines for the parsing (`ParseURI` ([585](#))) and construction (`EncodeURI` ([584](#))) 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..

33.2 Constants, types and variables

33.2.1 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` ([585](#)) call. It contains the contents of a URI, parsed out in it's various pieces.

33.3 Procedures and functions

33.3.1 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` ([585](#))

33.3.2 FilenameToURI

Synopsis: Construct a URI from a filename

Declaration: `function FilenameToURI(const Filename: string) : string`

Visibility: default

Description: `FilenameToURI` takes `Filename` and constructs a `file: protocol` URI from it.

Errors: None.

See also: `URIToFilename` ([586](#))

33.3.3 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` ([585](#)), `URIToFileName` ([586](#))

33.3.4 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` ([584](#))

33.3.5 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 succesfully returned.

Errors: If no protocols are specified, the function returns `False`

33.3.6 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` ([586](#)), `FilenameToURI` ([585](#))

Chapter 34

Reference for unit 'zstream'

34.1 Used units

Table 34.1: Used units by unit 'zstream'

Name	Page
Classes	??
gzio	??
System	??
zbase	??

34.2 Overview

The `ZStream` unit implements a `TStream` (??) descendent (`TCompressionStream` (588)) 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` (591)) and decompresses it with the inflate algorithm.

34.3 Constants, types and variables

34.3.1 Types

```
Tcompressionlevel = (clnone, clfastest, cldefault, clmax)
```

Table 34.2: Enumeration values for type `Tcompressionlevel`

Value	Explanation
<code>cldefault</code>	Use default compression
<code>clfastest</code>	Use fast (but less) compression.
<code>clmax</code>	Use maximum compression
<code>clnone</code>	Do not use compression, just copy data.

Compression level for the deflate algorithm

`Tgzopenmode = (gzopenread, gzopenwrite)`

Table 34.3: Enumeration values for type `Tgzopenmode`

Value	Explanation
<code>gzopenread</code>	Open file for reading
<code>gzopenwrite</code>	Open file for writing

Open mode for gzip file.

34.4 Ecompressionerror

34.4.1 Description

`ECompressionError` is the exception class used by the `TCompressionStream` (588) class.

34.5 Edecompressionerror

34.5.1 Description

`EDecompressionError` is the exception class used by the `TDecompressionStream` (591) class.

34.6 Egzfileerror

34.6.1 Description

`Egzfileerror` is the exception class used to report errors by the `Tgzfilestream` (593) class.

See also: `Tgzfilestream` (593)

34.7 Ezliberror

34.7.1 Description

Errors which occur in the `zstream` unit are signaled by raising an `EZLibError` exception descendant.

34.8 Tcompressionstream

34.8.1 Description

`TCompressionStream`

34.8.2 Method overview

Page	Property	Description
589	<code>create</code>	Create a new instance of the compression stream.
589	<code>destroy</code>	Flushe data to the output stream and destroys the compression stream.
590	<code>flush</code>	Flush remaining data to the target stream
590	<code>get_compressionrate</code>	Get the current compression rate
589	<code>write</code>	Write data to the stream

34.8.3 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` (??)

34.8.4 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` (??)

34.8.5 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 comresseses (deflates) them. The compressed result is written to the output stream.

Errors: If an error occurs, an `ECompressionError` ([588](#)) exception is raised.

See also: `Read` (??), `Seek` (??)

34.8.6 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.

34.8.7 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.

34.9 Tcustomzlibstream

34.9.1 Description

`TCustomZlibStream` serves as the ancestor class for the `TCompressionStream` (588) and `TDecompressionStream` (591) classes.

It introduces support for a progress handler, and stores the input or output stream.

34.9.2 Method overview

Page	Property	Description
590	<code>create</code>	Create a new instance of <code>TCustomZlibStream</code>
591	<code>destroy</code>	Clear up instance

34.9.3 Tcustomzlibstream.create

Synopsis: Create a new instance of `TCustomZlibStream`

Declaration: `constructor create(stream: TStream)`

Visibility: `public`

Description: `Create` creates a new instance of `TCustomZlibStream`. It stores a reference to the input/output stream, and initializes the deflate compression mechanism so they can be used by the descendents.

See also: `TCompressionStream` (588), `TDecompressionStream` (591)

34.9.4 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` (??)

34.10 Tdecompressionstream

34.10.1 Description

`TDecompressionStream` performs the inverse operation of `TCompressionStream` (588). A read operation reads data from an input stream and decompresses (inflates) the data it as it goes along.

The decompression stream reads it's compressed data from a stream with deflated data. This data can be created e.g. with a `TCompressionStream` (588) compression stream.

See also: `TCompressionStream` (588)

34.10.2 Method overview

Page	Property	Description
591	<code>create</code>	Creates a new instance of the <code>TDecompressionStream</code> stream
592	<code>destroy</code>	Destroys the <code>TDecompressionStream</code> instance
593	<code>get_compressionrate</code>	Get the current compression rate
592	<code>read</code>	Read data from the compressed stream
592	<code>seek</code>	Move stream position to a certain location in the stream.

34.10.3 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` (??)

34.10.4 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` (??)

34.10.5 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` (588) exception is raised.

See also: `Write` (??)

34.10.6 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` (588) exception is raised if the stream does not allow the requested seek operation.

See also: `Read` (??)

34.10.7 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.

34.11 TGZFileStream

34.11.1 Description

`TGZFileStream` can be used to read data from a gzip file, or to write data to a gzip file.

See also: `TCompressionStream` (588), `TDeCompressionStream` (591)

34.11.2 Method overview

Page	Property	Description
593	<code>create</code>	Create a new instance of <code>TGZFileStream</code>
595	<code>destroy</code>	Removes <code>TGZFileStream</code> instance
594	<code>read</code>	Read data from the compressed file
594	<code>seek</code>	Set the position in the compressed stream.
594	<code>write</code>	Write data to be compressed

34.11.3 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` (??) 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` (??) method will be stored in the file in compressed (deflated) form.

Errors: If the file is not found, an `EZlibError` (588) exception is raised.

See also: `Destroy` (??), `TGZOpenMode` (588)

34.11.4 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 (??)`, `Write (??)`, `Seek (??)`

34.11.5 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 (588)` exception is raised.

See also: `Create (??)`, `Read (??)`, `Seek (??)`

34.11.6 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 (??)` for a list of possibilities.

Errors: In case an impossible combination is asked, an `EZlibError (588)` exception is raised.

See also: `TDecompressionStream.Seek (??)`

34.11.7 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 (??)