Free Component Library (FCL): Reference guide.

Reference guide for FCL units.
Document version 2.1
July 2012

Michaël Van Canneyt

Contents

0.1 Overview 54

1 Reference for unit 'ascii85' 55

1.1 Used units 55
1.2 Overview 55
1.3 Constants, types and variables 55
1.3.1 Types 55
1.4 TASCII85DecoderStream 55
1.4.1 Description 55
1.4.2 Method overview 56
1.4.3 Property overview 56
1.4.4 TASCII85DecoderStream.Create 56
1.4.5 TASCII85DecoderStream.Decode 56
1.4.6 TASCII85DecoderStream.Close 57
1.4.7 TASCII85DecoderStream.ClosedP 57
1.4.8 TASCII85DecoderStream.Destroy 57
1.4.9 TASCII85DecoderStream.Read 57
1.4.10 TASCII85DecoderStream.Seek 58
1.4.11 TASCII85DecoderStream.BExpectBoundary 58
1.5 TASCII85EncoderStream 58
1.5.1 Method overview 58
1.5.2 Property overview 58
1.5.3 TASCII85EncoderStream.Create 58
1.5.4 TASCII85EncoderStream.Destroy 59
1.5.5 TASCII85EncoderStream.Write 59
1.5.6 TASCII85EncoderStream.Width 59
1.5.7 TASCII85EncoderStream.Boundary 59
1.6 TASCII85RingBuffer 59
1.6.1 Description 59
1.6.2 Method overview 59
1.6.3 Property overview 59

1.6.4 TASCII85RingBuffer.Write . . . 59
1.6.5 TASCII85RingBuffer.Read . . . 60
1.6.6 TASCII85RingBuffer.FillCount . . . 60
1.6.7 TASCII85RingBuffer.Size . . . 60

2 Reference for unit 'AVL_Tree' 61
2.1 Used units . . . 61
2.2 Overview . . . 61
2.3 TAVLTree . . . 61
2.3.1 Description . . . 61
2.3.2 Method overview . . . 62
2.3.3 Property overview . . . 62
2.3.4 TAVLTree.Find . . . 62
2.3.5 TAVLTree.FindKey . . . 63
2.3.6 TAVLTree.FindSuccessor . . . 63
2.3.7 TAVLTree.FindPrecessor . . . 63
2.3.8 TAVLTree.FindLowest . . . 63
2.3.9 TAVLTree.FindHighest . . . 64
2.3.10 TAVLTree.FindNearest . . . 64
2.3.11 TAVLTree.FindPointer . . . 64
2.3.12 TAVLTree.FindLeftMost . . . 64
2.3.13 TAVLTree.FindRightMost . . . 65
2.3.14 TAVLTree.FindLeftMostKey . . . 65
2.3.15 TAVLTree.FindRightMostKey . . . 65
2.3.16 TAVLTree.FindLeftMostSameKey . . . 65
2.3.17 TAVLTree.FindRightMostSameKey . . . 66
2.3.18 TAVLTree.Add . . . 66
2.3.19 TAVLTree.Delete . . . 66
2.3.20 TAVLTree.Remove . . . 66
2.3.21 TAVLTree.RemovePointer . . . 67
2.3.22 TAVLTree.MoveDataLeftMost . . . 67
2.3.23 TAVLTree.MoveDataRightMost . . . 67
2.3.24 TAVLTree.Clear . . . 67
2.3.25 TAVLTree.FreeAndClear . . . 68
2.3.26 TAVLTree.FreeAndDelete . . . 68
2.3.27 TAVLTree.ConsistencyCheck . . . 68
2.3.28 TAVLTree.WriteReportToStream . . . 68
2.3.29 TAVLTree.ReportAsString . . . 69
2.3.30 TAVLTree.SetNodeManager . . . 69
2.3.31 TAVLTree.Create . . . 69

2.3.32 TAVLTree.Destroy . . . 69
2.3.33 TAVLTree.OnCompare . . . 69
2.3.34 TAVLTree.Count . . . 70
2.4 TAVLTreeNode . . . 70
2.4.1 Description . . . 70
2.4.2 Method overview . . . 70
2.4.3 TAVLTreeNode.Clear . . . 70
2.4.4 TAVLTreeNode.TreeDepth . . . 70
2.5 TAVLTreeNodeMemManager . . . 70
2.5.1 Description . . . 70
2.5.2 Method overview . . . 71
2.5.3 Property overview . . . 71
2.5.4 TAVLTreeNodeMemManager.DisposeNode . . . 71
2.5.5 TAVLTreeNodeMemManager.NewNode . . . 71
2.5.6 TAVLTreeNodeMemManager.Clear . . . 71
2.5.7 TAVLTreeNodeMemManager.Create . . . 72
2.5.8 TAVLTreeNodeMemManager.Destroy . . . 72
2.5.9 TAVLTreeNodeMemManager.MinimumFreeNode . . . 72
2.5.10 TAVLTreeNodeMemManager.MaximumFreeNodeRatio . . . 72
2.5.11 TAVLTreeNodeMemManager.Count . . . 73
2.6 TBaseAVLTreeNodeManager . . . 73
2.6.1 Method overview . . . 73
2.6.2 TBaseAVLTreeNodeManager.DisposeNode . . . 73
2.6.3 TBaseAVLTreeNodeManager.NewNode . . . 73

3 Reference for unit 'base64' 74
3.1 Used units . . . 74
3.2 Overview . . . 74
3.3 Constants, types and variables . . . 74
3.3.1 Types . . . 74
3.4 EBase64DecodingException . . . 75
3.4.1 Description . . . 75
3.5 TBase64DecodingStream . . . 75
3.5.1 Description . . . 75
3.5.2 Method overview . . . 75
3.5.3 Property overview . . . 75
3.5.4 TBase64DecodingStream.Create . . . 75
3.5.5 TBase64DecodingStream.Reset . . . 76
3.5.6 TBase64DecodingStream.Read . . . 76
3.5.7 TBase64DecodingStream.Seek . . . 76

3.5.8 TBase64DecodingStream.EOF . . . 77
3.5.9 TBase64DecodingStream.Mode . . . 77
3.6 TBase64EncodingStream . . . 77
3.6.1 Description . . . 77
3.6.2 Method overview . . . 77
3.6.3 TBase64EncodingStream.Create . . . 77
3.6.4 TBase64EncodingStream.Destroy . . . 78
3.6.5 TBase64EncodingStream.Write . . . 78
3.6.6 TBase64EncodingStream.Seek . . . 78

4 Reference for unit 'BlowFish' 79
4.1 Used units . . . 79
4.2 Overview . . . 79
4.3 Constants, types and variables . . . 79
4.3.1 Constants . . . 79
4.3.2 Types . . . 79
4.4 EBlowFishError . . . 80
4.4.1 Description . . . 80
4.5 TBlowFish . . . 80
4.5.1 Description . . . 80
4.5.2 Method overview . . . 80
4.5.3 TBlowFish.Create . . . 80
4.5.4 TBlowFish.Encrypt . . . 80
4.5.5 TBlowFish.Decrypt . . . 81
4.6 TBlowFishDeCryptStream . . . 81
4.6.1 Description . . . 81
4.6.2 Method overview . . . 81
4.6.3 TBlowFishDeCryptStream.Read . . . 81
4.6.4 TBlowFishDeCryptStream.Seek . . . 82
4.7 TBlowFishEncryptStream . . . 82
4.7.1 Description . . . 82
4.7.2 Method overview . . . 82
4.7.3 TBlowFishEncryptStream.Destroy . . . 82
4.7.4 TBlowFishEncryptStream.Write . . . 82
4.7.5 TBlowFishEncryptStream.Seek . . . 83
4.7.6 TBlowFishEncryptStream.Flush . . . 83
4.8 TBlowFishStream . . . 83
4.8.1 Description . . . 83
4.8.2 Method overview . . . 84
4.8.3 Property overview . . . 84

4.8.4 TBlowFishStream.Create . . . 84
4.8.5 TBlowFishStream.Destroy . . . 84
4.8.6 TBlowFishStream.BlowFish . . . 84

5 Reference for unit 'bufstream' 86
5.1 Used units . . . 86
5.2 Overview . . . 86
5.3 Constants, types and variables . . . 86
5.3.1 Constants . . . 86
5.4 TBufStream . . . 86
5.4.1 Description . . . 86
5.4.2 Method overview . . . 87
5.4.3 Property overview . . . 87
5.4.4 TBufStream.Create . . . 87
5.4.5 TBufStream.Destroy . . . 87
5.4.6 TBufStream.Buffer . . . 87
5.4.7 TBufStream.Capacity . . . 88
5.4.8 TBufStream.BufferPos . . . 88
5.4.9 TBufStream.BufferSize . . . 88
5.5 TReadBufStream . . . 89
5.5.1 Description . . . 89
5.5.2 Method overview . . . 89
5.5.3 TReadBufStream.Seek . . . 89
5.5.4 TReadBufStream.Read . . . 89
5.6 TWriteBufStream . . . 89
5.6.1 Description . . . 89
5.6.2 Method overview . . . 90
5.6.3 TWriteBufStream.Destroy . . . 90
5.6.4 TWriteBufStream.Seek . . . 90
5.6.5 TWriteBufStream.Write . . . 90

6 Reference for unit 'CacheCls' 91
6.1 Used units . . . 91
6.2 Overview . . . 91
6.3 Constants, types and variables . . . 91
6.3.1 Resource strings . . . 91
6.3.2 Types . . . 91
6.4 ECacheError . . . 92
6.4.1 Description . . . 92
6.5 TCache . . . 92
6.5.1 Description . . . 92

6.5.2 Method overview . . . 93
6.5.3 Property overview . . . 93
6.5.4 TCache.Create . . . 93
6.5.5 TCache.Destroy . . . 93
6.5.6 TCache.Add . . . 93
6.5.7 TCache.AddNew . . . 94
6.5.8 TCache.FindSlot . . . 94
6.5.9 TCache.IndexOf . . . 94
6.5.10 TCache.Remove . . . 95
6.5.11 TCache.Data . . . 95
6.5.12 TCache.MRUSlot . . . 95
6.5.13 TCache.LRUSlot . . . 96
6.5.14 TCache.SlotCount . . . 96
6.5.15 TCache.Slots . . . 96
6.5.16 TCache.OnIsDataEqual . . . 96
6.5.17 TCache.OnFreeSlot . . . 97

7 Reference for unit 'contnrs' 98
7.1 Used units . . . 98
7.2 Overview . . . 98
7.3 Constants, types and variables . . . 98
7.3.1 Constants . . . 98
7.3.2 Types . . . 99
7.4 Procedures and functions . . . 102
7.4.1 RSHash . . . 102
7.5 EDuplicate . . . 102
7.5.1 Description . . . 102
7.6 EKeyNotFound . . . 102
7.6.1 Description . . . 102
7.7 TBucketList . . . 102
7.7.1 Description . . . 102
7.7.2 Method overview . . . 103
7.7.3 TBucketList.Create . . . 103
7.8 TClassList . . . 103
7.8.1 Description . . . 103
7.8.2 Method overview . . . 103
7.8.3 Property overview . . . 103
7.8.4 TClassList.Add . . . 103
7.8.5 TClassList.Extract . . . 104
7.8.6 TClassList.Remove . . . 104

7.8.7 TClassList.IndexOf . . . 104
7.8.8 TClassList.First . . . 105
7.8.9 TClassList.Last . . . 105
7.8.10 TClassList.Insert . . . 105
7.8.11 TClassList.Items . . . 105
7.9 TComponentList . . . 106
7.9.1 Description . . . 106
7.9.2 Method overview . . . 106
7.9.3 Property overview . . . 106
7.9.4 TComponentList.Destroy . . . 106
7.9.5 TComponentList.Add . . . 106
7.9.6 TComponentList.Extract . . . 107
7.9.7 TComponentList.Remove . . . 107
7.9.8 TComponentList.IndexOf . . . 107
7.9.9 TComponentList.First . . . 108
7.9.10 TComponentList.Last . . . 108
7.9.11 TComponentList.Insert . . . 108
7.9.12 TComponentList.Items . . . 108
7.10 TCustomBucketList . . . 109
7.10.1 Description . . . 109
7.10.2 Method overview . . . 109
7.10.3 Property overview . . . 109
7.10.4 TCustomBucketList.Destroy . . . 109
7.10.5 TCustomBucketList.Clear . . . 109
7.10.6 TCustomBucketList.Add . . . 110
7.10.7 TCustomBucketList.Assign . . . 110
7.10.8 TCustomBucketList.Exists . . . 110
7.10.9 TCustomBucketList.Find . . . 110
7.10.10 TCustomBucketList.ForEach . . . 111
7.10.11 TCustomBucketList.Remove . . . 111
7.10.12 TCustomBucketList.Data . . . 111
7.11 TFPCustomHashTable . . . 111
7.11.1 Description . . . 111
7.11.2 Method overview . . . 112
7.11.3 Property overview . . . 112
7.11.4 TFPCustomHashTable.Create . . . 112
7.11.5 TFPCustomHashTable.CreateWith . . . 112
7.11.6 TFPCustomHashTable.Destroy . . . 113
7.11.7 TFPCustomHashTable.ChangeTableSize . . . 113
7.11.8 TFPCustomHashTable.Clear . . . 113

7.11.9 TFPCustomHashTable.Delete . . . 114
7.11.10 TFPCustomHashTable.Find . . . 114
7.11.11 TFPCustomHashTable.IsEmpty . . . 114
7.11.12 TFPCustomHashTable.HashFunction . . . 114
7.11.13 TFPCustomHashTable.Count . . . 115
7.11.14 TFPCustomHashTable.HashTableSize . . . 115
7.11.15 TFPCustomHashTable.HashTable . . . 115
7.11.16 TFPCustomHashTable.VoidSlots . . . 116
7.11.17 TFPCustomHashTable.LoadFactor . . . 116
7.11.18 TFPCustomHashTable.AVGChainLen . . . 116
7.11.19 TFPCustomHashTable.MaxChainLength . . . 116
7.11.20 TFPCustomHashTable.NumberOfCollisions . . . 117
7.11.21 TFPCustomHashTable.Density . . . 117
7.12 TFPDataHashTable . . . 117
7.12.1 Description . . . 117
7.12.2 Method overview . . . 117
7.12.3 Property overview . . . 118
7.12.4 TFPDataHashTable.Add . . . 118
7.12.5 TFPDataHashTable.Items . . . 118
7.13 TFPHashList . . . 118
7.13.1 Description . . . 118
7.13.2 Method overview . . . 119
7.13.3 Property overview . . . 119
7.13.4 TFPHashList.Create . . . 119
7.13.5 TFPHashList.Destroy . . . 119
7.13.6 TFPHashList.Add . . . 120
7.13.7 TFPHashList.Clear . . . 120
7.13.8 TFPHashList.NameOfIndex . . . 120
7.13.9 TFPHashList.HashOfIndex . . . 120
7.13.10 TFPHashList.GetNextCollision . . . 121
7.13.11 TFPHashList.Delete . . . 121
7.13.12 TFPHashList.Error . . . 121
7.13.13 TFPHashList.Expand . . . 121
7.13.14 TFPHashList.Extract . . . 122
7.13.15 TFPHashList.IndexOf . . . 122
7.13.16 TFPHashList.Find . . . 122
7.13.17 TFPHashList.FindIndexOf . . . 122
7.13.18 TFPHashList.FindWithHash . . . 123
7.13.19 TFPHashList.Rename . . . 123
7.13.20 TFPHashList.Remove . . . 123

7.13.21 TFPHashList.Pack . . . 123
7.13.22 TFPHashList.ShowStatistics . . . 124
7.13.23 TFPHashList.ForEachCall . . . 124
7.13.24 TFPHashList.Capacity . . . 124
7.13.25 TFPHashList.Count . . . 124
7.13.26 TFPHashList.Items . . . 125
7.13.27 TFPHashList.List . . . 125
7.13.28 TFPHashList.Strs . . . 125
7.14 TFPHashObject . . . 125
7.14.1 Description . . . 125
7.14.2 Method overview . . . 126
7.14.3 Property overview . . . 126
7.14.4 TFPHashObject.CreateNotOwned . . . 126
7.14.5 TFPHashObject.Create . . . 126
7.14.6 TFPHashObject.ChangeOwner . . . 126
7.14.7 TFPHashObject.ChangeOwnerAndName . . . 127
7.14.8 TFPHashObject.Rename . . . 127
7.14.9 TFPHashObject.Name . . . 127
7.14.10 TFPHashObject.Hash . . . 127
7.15 TFPHashObjectList . . . 128
7.15.1 Method overview . . . 128
7.15.2 Property overview . . . 128
7.15.3 TFPHashObjectList.Create . . . 128
7.15.4 TFPHashObjectList.Destroy . . . 128
7.15.5 TFPHashObjectList.Clear . . . 129
7.15.6 TFPHashObjectList.Add . . . 129
7.15.7 TFPHashObjectList.NameOfIndex . . . 129
7.15.8 TFPHashObjectList.HashOfIndex . . . 130
7.15.9 TFPHashObjectList.GetNextCollision . . . 130
7.15.10 TFPHashObjectList.Delete . . . 130
7.15.11 TFPHashObjectList.Expand . . . 130
7.15.12 TFPHashObjectList.Extract . . . 131
7.15.13 TFPHashObjectList.Remove . . . 131
7.15.14 TFPHashObjectList.IndexOf . . . 131
7.15.15 TFPHashObjectList.Find . . . 131
7.15.16 TFPHashObjectList.FindIndexOf . . . 132
7.15.17 TFPHashObjectList.FindWithHash . . . 132
7.15.18 TFPHashObjectList.Rename . . . 132
7.15.19 TFPHashObjectList.FindInstanceOf . . . 132
7.15.20 TFPHashObjectList.Pack . . . 133

7.15.21 TFPHashObjectList.ShowStatistics . . . 133
7.15.22 TFPHashObjectList.ForEachCall . . . 133
7.15.23 TFPHashObjectList.Capacity . . . 133
7.15.24 TFPHashObjectList.Count . . . 134
7.15.25 TFPHashObjectList.OwnsObjects . . . 134
7.15.26 TFPHashObjectList.Items . . . 134
7.15.27 TFPHashObjectList.List . . . 134
7.16 TFPObjectHashTable . . . 135
7.16.1 Description . . . 135
7.16.2 Method overview . . . 135
7.16.3 Property overview . . . 135
7.16.4 TFPObjectHashTable.Create . . . 135
7.16.5 TFPObjectHashTable.CreateWith . . . 135
7.16.6 TFPObjectHashTable.Add . . . 136
7.16.7 TFPObjectHashTable.Items . . . 136
7.16.8 TFPObjectHashTable.OwnsObjects . . . 136
7.17 TFPObjectList . . . 137
7.17.1 Description . . . 137
7.17.2 Method overview . . . 137
7.17.3 Property overview . . . 137
7.17.4 TFPObjectList.Create . . . 137
7.17.5 TFPObjectList.Destroy . . . 138
7.17.6 TFPObjectList.Clear . . . 138
7.17.7 TFPObjectList.Add . . . 138
7.17.8 TFPObjectList.Delete . . . 138
7.17.9 TFPObjectList.Exchange . . . 139
7.17.10 TFPObjectList.Expand . . . 139
7.17.11 TFPObjectList.Extract . . . 139
7.17.12 TFPObjectList.Remove . . . 140
7.17.13 TFPObjectList.IndexOf . . . 140
7.17.14 TFPObjectList.FindInstanceOf . . . 140
7.17.15 TFPObjectList.Insert . . . 140
7.17.16 TFPObjectList.First . . . 141
7.17.17 TFPObjectList.Last . . . 141
7.17.18 TFPObjectList.Move . . . 141
7.17.19 TFPObjectList.Assign . . . 142
7.17.20 TFPObjectList.Pack . . . 142
7.17.21 TFPObjectList.Sort . . . 142
7.17.22 TFPObjectList.ForEachCall . . . 142
7.17.23 TFPObjectList.Capacity . . . 143

7.17.24 TFPObjectList.Count . . . 143
7.17.25 TFPObjectList.OwnsObjects . . . 143
7.17.26 TFPObjectList.Items . . . 144
7.17.27 TFPObjectList.List . . . 144
7.18 TFPStringHashTable . . . 144
7.18.1 Description . . . 144
7.18.2 Method overview . . . 144
7.18.3 Property overview . . . 144
7.18.4 TFPStringHashTable.Add . . . 144
7.18.5 TFPStringHashTable.Items . . . 145
7.19 THTCustomNode . . . 145
7.19.1 Description . . . 145
7.19.2 Method overview . . . 145
7.19.3 Property overview . . . 145
7.19.4 THTCustomNode.CreateWith . . . 145
7.19.5 THTCustomNode.HasKey . . . 146
7.19.6 THTCustomNode.Key . . . 146
7.20 THTDataNode . . . 146
7.20.1 Description . . . 146
7.20.2 Property overview . . . 146
7.20.3 THTDataNode.Data . . . 146
7.21 THTObjectNode . . . 147
7.21.1 Description . . . 147
7.21.2 Property overview . . . 147
7.21.3 THTObjectNode.Data . . . 147
7.22 THTOwnedObjectNode . . . 147
7.22.1 Description . . . 147
7.22.2 Method overview . . . 147
7.22.3 THTOwnedObjectNode.Destroy . . . 147
7.23 THTStringNode . . . 148
7.23.1 Description . . . 148
7.23.2 Property overview . . . 148
7.23.3 THTStringNode.Data . . . 148
7.24 TObjectBucketList . . . 148
7.24.1 Description . . . 148
7.24.2 Method overview . . . 148
7.24.3 Property overview . . . 148
7.24.4 TObjectBucketList.Add . . . 148
7.24.5 TObjectBucketList.Remove . . . 149
7.24.6 TObjectBucketList.Data . . . 149

7.25 TObjectList . . . 149
7.25.1 Description . . . 149
7.25.2 Method overview . . . 149
7.25.3 Property overview . . . 150
7.25.4 TObjectList.create . . . 150
7.25.5 TObjectList.Add . . . 150
7.25.6 TObjectList.Extract . . . 150
7.25.7 TObjectList.Remove . . . 151
7.25.8 TObjectList.IndexOf . . . 151
7.25.9 TObjectList.FindInstanceOf . . . 151
7.25.10 TObjectList.Insert . . . 152
7.25.11 TObjectList.First . . . 152
7.25.12 TObjectList.Last . . . 152
7.25.13 TObjectList.OwnsObjects . . . 152
7.25.14 TObjectList.Items . . . 153
7.26 TObjectQueue . . . 153
7.26.1 Method overview . . . 153
7.26.2 TObjectQueue.Push . . . 153
7.26.3 TObjectQueue.Pop . . . 153
7.26.4 TObjectQueue.Peek . . . 154
7.27 TObjectStack . . . 154
7.27.1 Description . . . 154
7.27.2 Method overview . . . 154
7.27.3 TObjectStack.Push . . . 154
7.27.4 TObjectStack.Pop . . . 154
7.27.5 TObjectStack.Peek . . . 155
7.28 TOrderedList . . . 155
7.28.1 Description . . . 155
7.28.2 Method overview . . . 155
7.28.3 TOrderedList.Create . . . 155
7.28.4 TOrderedList.Destroy . . . 155
7.28.5 TOrderedList.Count . . . 156
7.28.6 TOrderedList.AtLeast . . . 156
7.28.7 TOrderedList.Push . . . 156
7.28.8 TOrderedList.Pop . . . 157
7.28.9 TOrderedList.Peek . . . 157
7.29 TQueue . . . 157
7.29.1 Description . . . 157
7.30 TStack . . . 157
7.30.1 Description . . . 157

8 Reference for unit 'CustApp' 158
8.1 Used units . . . 158
8.2 Overview . . . 158
8.3 Constants, types and variables . . . 158
8.3.1 Types . . . 158
8.3.2 Variables . . . 159
8.4 TCustomApplication . . . 159
8.4.1 Description . . . 159
8.4.2 Method overview . . . 159
8.4.3 Property overview . . . 160
8.4.4 TCustomApplication.Create . . . 160
8.4.5 TCustomApplication.Destroy . . . 160
8.4.6 TCustomApplication.HandleException . . . 160
8.4.7 TCustomApplication.Initialize . . . 161
8.4.8 TCustomApplication.Run . . . 161
8.4.9 TCustomApplication.ShowException . . . 161
8.4.10 TCustomApplication.Terminate . . . 162
8.4.11 TCustomApplication.FindOptionIndex . . . 162
8.4.12 TCustomApplication.GetOptionValue . . . 162
8.4.13 TCustomApplication.HasOption . . . 163
8.4.14 TCustomApplication.CheckOptions . . . 163
8.4.15 TCustomApplication.GetEnvironmentList . . . 164
8.4.16 TCustomApplication.Log . . . 164
8.4.17 TCustomApplication.ExeName . . . 164
8.4.18 TCustomApplication.HelpFile . . . 165
8.4.19 TCustomApplication.Terminated . . . 165
8.4.20 TCustomApplication.Title . . . 165
8.4.21 TCustomApplication.OnException . . . 166
8.4.22 TCustomApplication.ConsoleApplication . . . 166
8.4.23 TCustomApplication.Location . . . 166
8.4.24 TCustomApplication.Params . . . 166
8.4.25 TCustomApplication.ParamCount . . . 167
8.4.26 TCustomApplication.EnvironmentVariable . . . 167
8.4.27 TCustomApplication.OptionChar . . . 167
8.4.28 TCustomApplication.CaseSensitiveOptions . . . 168
8.4.29 TCustomApplication.StopOnException . . . 168

9 Reference for unit 'daemonapp' 169
9.1 Used units . . . 169
9.2 Overview . . . 169

9.3 Daemon application architecture . . . 170
9.4 Constants, types and variables . . . 170
9.4.1 Resource strings . . . 170
9.4.2 Types . . . 171
9.4.3 Variables . . . 173
9.5 Procedures and functions . . . 174
9.5.1 Application . . . 174
9.5.2 DaemonError . . . 174
9.5.3 RegisterDaemonApplicationClass . . . 174
9.5.4 RegisterDaemonClass . . . 175
9.5.5 RegisterDaemonMapper . . . 175
9.6 EDaemon . . . 175
9.6.1 Description . . . 175
9.7 TCustomDaemon . . . 175
9.7.1 Description . . . 175
9.7.2 Method overview . . . 175
9.7.3 Property overview . . . 176
9.7.4 TCustomDaemon.LogMessage . . . 176
9.7.5 TCustomDaemon.ReportStatus . . . 176
9.7.6 TCustomDaemon.Definition . . . 176
9.7.7 TCustomDaemon.DaemonThread . . . 177
9.7.8 TCustomDaemon.Controller . . . 177
9.7.9 TCustomDaemon.Status . . . 177
9.7.10 TCustomDaemon.Logger . . . 178
9.8 TCustomDaemonApplication . . . 178
9.8.1 Description . . . 178
9.8.2 Method overview . . . 178
9.8.3 Property overview . . . 178
9.8.4 TCustomDaemonApplication.ShowException . . . 178
9.8.5 TCustomDaemonApplication.CreateDaemon . . . 179
9.8.6 TCustomDaemonApplication.StopDaemons . . . 179
9.8.7 TCustomDaemonApplication.InstallDaemons . . . 179
9.8.8 TCustomDaemonApplication.RunDaemons . . . 179
9.8.9 TCustomDaemonApplication.UnInstallDaemons . . . 180
9.8.10 TCustomDaemonApplication.CreateForm . . . 180
9.8.11 TCustomDaemonApplication.OnRun . . . 180
9.8.12 TCustomDaemonApplication.Logger . . . 180
9.8.13 TCustomDaemonApplication.GUIMainLoop . . . 181
9.8.14 TCustomDaemonApplication.GuiHandle . . . 181
9.8.15 TCustomDaemonApplication.RunMode . . . 181

9.9 TCustomDaemonMapper . . . 182
9.9.1 Description . . . 182
9.9.2 Method overview . . . 182
9.9.3 Property overview . . . 182
9.9.4 TCustomDaemonMapper.Create . . . 182
9.9.5 TCustomDaemonMapper.Destroy . . . 182
9.9.6 TCustomDaemonMapper.DaemonDefs . . . 183
9.9.7 TCustomDaemonMapper.OnCreate . . . 183
9.9.8 TCustomDaemonMapper.OnDestroy . . . 183
9.9.9 TCustomDaemonMapper.OnRun . . . 183
9.9.10 TCustomDaemonMapper.OnInstall . . . 184
9.9.11 TCustomDaemonMapper.OnUnInstall . . . 184
9.10 TDaemon . . . 184
9.10.1 Description . . . 184
9.10.2 Property overview . . . 185
9.10.3 TDaemon.Definition . . . 185
9.10.4 TDaemon.Status . . . 185
9.10.5 TDaemon.OnStart . . . 185
9.10.6 TDaemon.OnStop . . . 186
9.10.7 TDaemon.OnPause . . . 186
9.10.8 TDaemon.OnContinue . . . 186
9.10.9 TDaemon.OnShutDown . . . 187
9.10.10 TDaemon.OnExecute . . . 187
9.10.11 TDaemon.BeforeInstall . . . 187
9.10.12 TDaemon.AfterInstall . . . 188
9.10.13 TDaemon.BeforeUnInstall . . . 188
9.10.14 TDaemon.AfterUnInstall . . . 188
9.10.15 TDaemon.OnControlCode . . . 188
9.11 TDaemonApplication . . . 189
9.11.1 Description . . . 189
9.12 TDaemonController . . . 189
9.12.1 Description . . . 189
9.12.2 Method overview . . . 189
9.12.3 Property overview . . . 189
9.12.4 TDaemonController.Create . . . 189
9.12.5 TDaemonController.Destroy . . . 190
9.12.6 TDaemonController.StartService . . . 190
9.12.7 TDaemonController.Main . . . 190
9.12.8 TDaemonController.Controller . . . 190
9.12.9 TDaemonController.ReportStatus . . . 191

9.12.10 TDaemonController.Daemon . . . 191
9.12.11 TDaemonController.Params . . . 191
9.12.12 TDaemonController.LastStatus . . . 191
9.12.13 TDaemonController.CheckPoint . . . 192
9.13 TDaemonDef . . . 192
9.13.1 Description . . . 192
9.13.2 Method overview . . . 192
9.13.3 Property overview . . . 192
9.13.4 TDaemonDef.Create . . . 192
9.13.5 TDaemonDef.Destroy . . . 193
9.13.6 TDaemonDef.DaemonClass . . . 193
9.13.7 TDaemonDef.Instance . . . 193
9.13.8 TDaemonDef.DaemonClassName . . . 193
9.13.9 TDaemonDef.Name . . . 194
9.13.10 TDaemonDef.Description . . . 194
9.13.11 TDaemonDef.DisplayName . . . 194
9.13.12 TDaemonDef.RunArguments . . . 194
9.13.13 TDaemonDef.Options . . . 195
9.13.14 TDaemonDef.Enabled . . . 195
9.13.15 TDaemonDef.WinBindings . . . 195
9.13.16 TDaemonDef.OnCreateInstance . . . 195
9.13.17 TDaemonDef.LogStatusReport . . . 196
9.14 TDaemonDefs . . . 196
9.14.1 Description . . . 196
9.14.2 Method overview . . . 196
9.14.3 Property overview . . . 196
9.14.4 TDaemonDefs.Create . . . 196
9.14.5 TDaemonDefs.IndexOfDaemonDef . . . 197
9.14.6 TDaemonDefs.FindDaemonDef . . . 197
9.14.7 TDaemonDefs.DaemonDefByName . . . 197
9.14.8 TDaemonDefs.Daemons . . . 197
9.15 TDaemonMapper . . . 198
9.15.1 Description . . . 198
9.15.2 Method overview . . . 198
9.15.3 TDaemonMapper.Create . . . 198
9.15.4 TDaemonMapper.CreateNew . . . 198
9.16 TDaemonThread . . . 198
9.16.1 Description . . . 198
9.16.2 Method overview . . . 199
9.16.3 Property overview . . . 199

9.16.4 TDaemonThread.Create . . . 199
9.16.5 TDaemonThread.Execute . . . 199
9.16.6 TDaemonThread.CheckControlMessage . . . 199
9.16.7 TDaemonThread.StopDaemon . . . 200
9.16.8 TDaemonThread.PauseDaemon . . . 200
9.16.9 TDaemonThread.ContinueDaemon . . . 200
9.16.10 TDaemonThread.ShutDownDaemon . . . 200
9.16.11 TDaemonThread.InterrogateDaemon . . . 201
9.16.12 TDaemonThread.Daemon . . . 201
9.17 TDependencies . . . 201
9.17.1 Description . . . 201
9.17.2 Method overview . . . 201
9.17.3 Property overview . . . 201
9.17.4 TDependencies.Create . . . 201
9.17.5 TDependencies.Items . . . 202
9.18 TDependency . . . 202
9.18.1 Description . . . 202
9.18.2 Method overview . . . 202
9.18.3 Property overview . . . 202
9.18.4 TDependency.Assign . . . 202
9.18.5 TDependency.Name . . . 202
9.18.6 TDependency.IsGroup . . . 203
9.19 TWinBindings . . . 203
9.19.1 Description . . . 203
9.19.2 Method overview . . . 203
9.19.3 Property overview . . . 203
9.19.4 TWinBindings.Create . . . 203
9.19.5 TWinBindings.Destroy . . . 204
9.19.6 TWinBindings.Assign . . . 204
9.19.7 TWinBindings.ErrCode . . . 204
9.19.8 TWinBindings.Win32ErrCode . . . 204
9.19.9 TWinBindings.Dependencies . . . 205
9.19.10 TWinBindings.GroupName . . . 205
9.19.11 TWinBindings.Password . . . 205
9.19.12 TWinBindings.UserName . . . 205
9.19.13 TWinBindings.StartType . . . 206
9.19.14 TWinBindings.WaitHint . . . 206
9.19.15 TWinBindings.IDTag . . . 206
9.19.16 TWinBindings.ServiceType . . . 207
9.19.17 TWinBindings.ErrorSeverity . . . 207

10 Reference for unit 'db' 208
10.1 Used units . . . 208
10.2 Overview . . . 208
10.3 Constants, types and variables . . . 208
10.3.1 Constants . . . 208
10.3.2 Types . . . 209
10.4 Procedures and functions . . . 218
10.4.1 BuffersEqual . . . 218
10.4.2 DatabaseError . . . 218
10.4.3 DatabaseErrorFmt . . . 219
10.4.4 DateTimeRecToDateTime . . . 219
10.4.5 DateTimeToDateTimeRec . . . 219
10.4.6 DisposeMem . . . 219
10.4.7 ExtractFieldName . . . 220
10.4.8 SkipComments . . . 220
10.5 EDatabaseError . . . 220
10.5.1 Description . . . 220
10.6 EUpdateError . . . 220
10.6.1 Description . . . 220
10.6.2 Method overview . . . 221
10.6.3 Property overview . . . 221
10.6.4 EUpdateError.Create . . . 221
10.6.5 EUpdateError.Destroy . . . 221
10.6.6 EUpdateError.Context . . . 221
10.6.7 EUpdateError.ErrorCode . . . 222
10.6.8 EUpdateError.OriginalExcaption . . . 222
10.6.9 EUpdateError.PreviousError . . . 222
10.7 TAutoIncField . . . 222
10.7.1 Description . . . 222
10.7.2 Method overview . . . 223
10.7.3 TAutoIncField.Create . . . 223
10.8 TBCDField . . . 223
10.8.1 Description . . . 223
10.8.2 Method overview . . . 223
10.8.3 Property overview . . . 223
10.8.4 TBCDField.Create . . . 223
10.8.5 TBCDField.CheckRange . . . 224
10.8.6 TBCDField.Value . . . 224
10.8.7 TBCDField.Precision . . . 224
10.8.8 TBCDField.Currency . . . 225

10.8.9 TBCDField.MaxValue . . . 225
10.8.10 TBCDField.MinValue . . . 225
10.8.11 TBCDField.Size . . . 226
10.9 TBinaryField . . . 226
10.9.1 Description . . . 226
10.9.2 Method overview . . . 226
10.9.3 Property overview . . . 226
10.9.4 TBinaryField.Create . . . 226
10.9.5 TBinaryField.Size . . . 226
10.10TBlobField . . . 227
10.10.1 Description . . . 227
10.10.2 Method overview . . . 227
10.10.3 Property overview . . . 227
10.10.4 TBlobField.Create . . . 227
10.10.5 TBlobField.Clear . . . 228
10.10.6 TBlobField.IsBlob . . . 228
10.10.7 TBlobField.LoadFromFile . . . 228
10.10.8 TBlobField.LoadFromStream . . . 228
10.10.9 TBlobField.SaveToFile . . . 229
10.10.10TBlobField.SaveToStream . . . 229
10.10.11TBlobField.SetFieldType . . . 229
10.10.12TBlobField.BlobSize . . . 229
10.10.13TBlobField.Modified . . . 230
10.10.14TBlobField.Value . . . 230
10.10.15TBlobField.Transliterate . . . 230
10.10.16TBlobField.BlobType . . . 230
10.10.17TBlobField.Size . . . 231
10.11TBooleanField . . . 231
10.11.1 Description . . . 231
10.11.2 Method overview . . . 231
10.11.3 Property overview . . . 231
10.11.4 TBooleanField.Create . . . 231
10.11.5 TBooleanField.Value . . . 232
10.11.6 TBooleanField.DisplayValues . . . 232
10.12TBytesField . . . 232
10.12.1 Description . . . 232
10.12.2 Method overview . . . 232
10.12.3 TBytesField.Create . . . 233
10.13TCheckConstraint . . . 233
10.13.1 Description . . . 233

10.13.2 Method overview . . . 233
10.13.3 Property overview . . . 233
10.13.4 TCheckConstraint.Assign . . . 233
10.13.5 TCheckConstraint.CustomConstraint . . . 234
10.13.6 TCheckConstraint.ErrorMessage . . . 234
10.13.7 TCheckConstraint.FromDictionary . . . 234
10.13.8 TCheckConstraint.ImportedConstraint . . . 234
10.14 TCheckConstraints . . . 235
10.14.1 Description . . . 235
10.14.2 Method overview . . . 235
10.14.3 Property overview . . . 235
10.14.4 TCheckConstraints.Create . . . 235
10.14.5 TCheckConstraints.Add . . . 235
10.14.6 TCheckConstraints.Items . . . 235
10.15 TCurrencyField . . . 236
10.15.1 Description . . . 236
10.15.2 Method overview . . . 236
10.15.3 Property overview . . . 236
10.15.4 TCurrencyField.Create . . . 236
10.15.5 TCurrencyField.Currency . . . 236
10.16 TCustomConnection . . . 237
10.16.1 Description . . . 237
10.16.2 Method overview . . . 237
10.16.3 Property overview . . . 237
10.16.4 TCustomConnection.Close . . . 237
10.16.5 TCustomConnection.Destroy . . . 237
10.16.6 TCustomConnection.Open . . . 238
10.16.7 TCustomConnection.DataSetCount . . . 238
10.16.8 TCustomConnection.DataSets . . . 238
10.16.9 TCustomConnection.Connected . . . 239
10.16.10 TCustomConnection.LoginPrompt . . . 239
10.16.11 TCustomConnection.AfterConnect . . . 239
10.16.12 TCustomConnection.AfterDisconnect . . . 239
10.16.13 TCustomConnection.BeforeConnect . . . 240
10.16.14 TCustomConnection.BeforeDisconnect . . . 240
10.16.15 TCustomConnection.OnLogin . . . 240
10.17 TDatabase . . . 241
10.17.1 Description . . . 241
10.17.2 Method overview . . . 241
10.17.3 Property overview . . . 241

10.17.4 TDatabase.Create . . . 241
10.17.5 TDatabase.Destroy . . . 242
10.17.6 TDatabase.CloseDataSets . . . 242
10.17.7 TDatabase.CloseTransactions . . . 242
10.17.8 TDatabase.StartTransaction . . . 242
10.17.9 TDatabase.EndTransaction . . . 243
10.17.10TDatabase.TransactionCount . . . 243
10.17.11TDatabase.Transactions . . . 243
10.17.12TDatabase.Directory . . . 243
10.17.13TDatabase.IsSQLBased . . . 244
10.17.14TDatabase.Connected . . . 244
10.17.15TDatabase.DatabaseName . . . 244
10.17.16TDatabase.KeepConnection . . . 244
10.17.17TDatabase.Params . . . 245
10.18TDataLink . . . 245
10.18.1 Description . . . 245
10.18.2 Method overview . . . 245
10.18.3 Property overview . . . 246
10.18.4 TDataLink.Create . . . 246
10.18.5 TDataLink.Destroy . . . 246
10.18.6 TDataLink.Edit . . . 246
10.18.7 TDataLink.UpdateRecord . . . 247
10.18.8 TDataLink.ExecuteAction . . . 247
10.18.9 TDataLink.UpdateAction . . . 247
10.18.10TDataLink.Active . . . 247
10.18.11TDataLink.ActiveRecord . . . 248
10.18.12TDataLink.BOF . . . 248
10.18.13TDataLink.BufferCount . . . 248
10.18.14TDataLink.DataSet . . . 249
10.18.15TDataLink.DataSource . . . 249
10.18.16TDataLink.DataSourceFixed . . . 249
10.18.17TDataLink.Editing . . . 249
10.18.18TDataLink.Eof . . . 250
10.18.19TDataLink.ReadOnly . . . 250
10.18.20TDataLink.RecordCount . . . 250
10.19TDataSet . . . 250
10.19.1 Description . . . 250
10.19.2 Method overview . . . 253
10.19.3 Property overview . . . 254
10.19.4 TDataSet.Create . . . 255

10.19.5 TDataSet.Destroy . . . 255
10.19.6 TDataSet.ActiveBuffer . . . 255
10.19.7 TDataSet.GetFieldData . . . 255
10.19.8 TDataSet.SetFieldData . . . 256
10.19.9 TDataSet.Append . . . 256
10.19.10TDataSet.AppendRecord . . . 256
10.19.11TDataSet.BookmarkValid . . . 257
10.19.12TDataSet.Cancel . . . 257
10.19.13TDataSet.CheckBrowseMode . . . 257
10.19.14TDataSet.ClearFields . . . 257
10.19.15TDataSet.Close . . . 258
10.19.16TDataSet.ControlsDisabled . . . 258
10.19.17TDataSet.CompareBookmarks . . . 258
10.19.18TDataSet.CreateBlobStream . . . 259
10.19.19TDataSet.CursorPosChanged . . . 259
10.19.20TDataSet.DataConvert . . . 259
10.19.21TDataSet.Delete . . . 259
10.19.22TDataSet.DisableControls . . . 260
10.19.23TDataSet.Edit . . . 260
10.19.24TDataSet.EnableControls . . . 261
10.19.25TDataSet.FieldByName . . . 261
10.19.26TDataSet.FindField . . . 261
10.19.27TDataSet.FindFirst . . . 262
10.19.28TDataSet.FindLast . . . 262
10.19.29TDataSet.FindNext . . . 262
10.19.30TDataSet.FindPrior . . . 262
10.19.31TDataSet.First . . . 263
10.19.32TDataSet.FreeBookmark . . . 263
10.19.33TDataSet.GetBookmark . . . 263
10.19.34TDataSet.GetCurrentRecord . . . 264
10.19.35TDataSet.GetFieldList . . . 264
10.19.36TDataSet.GetFieldNames . . . 264
10.19.37TDataSet.GotoBookmark . . . 264
10.19.38TDataSet.Insert . . . 265
10.19.39TDataSet.InsertRecord . . . 265
10.19.40TDataSet.IsEmpty . . . 265
10.19.41TDataSet.IsLinkedTo . . . 265
10.19.42TDataSet.IsSequenced . . . 266
10.19.43TDataSet.Last . . . 266
10.19.44TDataSet.Locate . . . 266

10.19.45TDataSet.Lookup . . . 267
10.19.46TDataSet.MoveBy . . . 267
10.19.47TDataSet.Next . . . 267
10.19.48TDataSet.Open . . . 268
10.19.49TDataSet.Post . . . 268
10.19.50TDataSet.Prior . . . 269
10.19.51TDataSet.Refresh . . . 269
10.19.52TDataSet.Resync . . . 269
10.19.53TDataSet.SetFields . . . 270
10.19.54TDataSet.Translate . . . 270
10.19.55TDataSet.UpdateCursorPos . . . 270
10.19.56TDataSet.UpdateRecord . . . 270
10.19.57TDataSet.UpdateStatus . . . 271
10.19.58TDataSet.BOF . . . 271
10.19.59TDataSet.Bookmark . . . 271
10.19.60TDataSet.CanModify . . . 272
10.19.61TDataSet.DataSource . . . 272
10.19.62TDataSet.DefaultFields . . . 273
10.19.63TDataSet.EOF . . . 273
10.19.64TDataSet.FieldCount . . . 274
10.19.65TDataSet.FieldDefs . . . 274
10.19.66TDataSet.Found . . . 274
10.19.67TDataSet.Modified . . . 275
10.19.68TDataSet.IsUniDirectional . . . 275
10.19.69TDataSet.RecordCount . . . 275
10.19.70TDataSet.RecNo . . . 276
10.19.71TDataSet.RecordSize . . . 276
10.19.72TDataSet.State . . . 276
10.19.73TDataSet.Fields . . . 277
10.19.74TDataSet.FieldValues . . . 277
10.19.75TDataSet.Filter . . . 277
10.19.76TDataSet.Filtered . . . 278
10.19.77TDataSet.FilterOptions . . . 278
10.19.78TDataSet.Active . . . 278
10.19.79TDataSet.AutoCalcFields . . . 279
10.19.80TDataSet.BeforeOpen . . . 279
10.19.81TDataSet.AfterOpen . . . 280
10.19.82TDataSet.BeforeClose . . . 280
10.19.83TDataSet.AfterClose . . . 280
10.19.84TDataSet.BeforeInsert . . . 280

10.19.85TDataSet.AfterInsert . . . 281
10.19.86TDataSet.BeforeEdit . . . 281
10.19.87TDataSet.AfterEdit . . . 281
10.19.88TDataSet.BeforePost . . . 282
10.19.89TDataSet.AfterPost . . . 282
10.19.90TDataSet.BeforeCancel . . . 282
10.19.91TDataSet.AfterCancel . . . 283
10.19.92TDataSet.BeforeDelete . . . 283
10.19.93TDataSet.AfterDelete . . . 283
10.19.94TDataSet.BeforeScroll . . . 283
10.19.95TDataSet.AfterScroll . . . 284
10.19.96TDataSet.BeforeRefresh . . . 284
10.19.97TDataSet.AfterRefresh . . . 284
10.19.98TDataSet.OnCalcFields . . . 285
10.19.99TDataSet.OnDeleteError . . . 285
10.19.100DataSet.OnEditError . . . 286
10.19.101DataSet.OnFilterRecord . . . 286
10.19.102DataSet.OnNewRecord . . . 286
10.19.103DataSet.OnPostError . . . 287
10.20TDataSource . . . 287
10.20.1 Description . . . 287
10.20.2 Method overview . . . 287
10.20.3 Property overview . . . 288
10.20.4 TDataSource.Create . . . 288
10.20.5 TDataSource.Destroy . . . 288
10.20.6 TDataSource.Edit . . . 288
10.20.7 TDataSource.IsLinkedTo . . . 289
10.20.8 TDataSource.State . . . 289
10.20.9 TDataSource.AutoEdit . . . 289
10.20.10TDataSource.DataSet . . . 289
10.20.11TDataSource.Enabled . . . 290
10.20.12TDataSource.OnStateChange . . . 290
10.20.13TDataSource.OnDataChange . . . 290
10.20.14TDataSource.OnUpdateData . . . 291
10.21TDateField . . . 291
10.21.1 Description . . . 291
10.21.2 Method overview . . . 291
10.21.3 TDateField.Create . . . 291
10.22TDateTimeField . . . 291
10.22.1 Description . . . 291

10.22.2 Method overview . . . 292
10.22.3 Property overview . . . 292
10.22.4 TDateTimeField.Create . . . 292
10.22.5 TDateTimeField.Value . . . 292
10.22.6 TDateTimeField.DisplayFormat . . . 292
10.23TDBDataset . . . 293
10.23.1 Description . . . 293
10.23.2 Method overview . . . 293
10.23.3 Property overview . . . 293
10.23.4 TDBDataset.destroy . . . 293
10.23.5 TDBDataset.DataBase . . . 293
10.23.6 TDBDataset.Transaction . . . 294
10.24TDBTransaction . . . 294
10.24.1 Description . . . 294
10.24.2 Method overview . . . 294
10.24.3 Property overview . . . 294
10.24.4 TDBTransaction.Create . . . 294
10.24.5 TDBTransaction.destroy . . . 295
10.24.6 TDBTransaction.CloseDataSets . . . 295
10.24.7 TDBTransaction.DataBase . . . 295
10.24.8 TDBTransaction.Active . . . 295
10.25TDefCollection . . . 296
10.25.1 Description . . . 296
10.25.2 Method overview . . . 296
10.25.3 Property overview . . . 296
10.25.4 TDefCollection.create . . . 296
10.25.5 TDefCollection.Find . . . 296
10.25.6 TDefCollection.GetItemNames . . . 297
10.25.7 TDefCollection.IndexOf . . . 297
10.25.8 TDefCollection.Dataset . . . 297
10.25.9 TDefCollection.Updated . . . 297
10.26TDetailDataLink . . . 298
10.26.1 Description . . . 298
10.26.2 Property overview . . . 298
10.26.3 TDetailDataLink.DetailDataSet . . . 298
10.27TField . . . 298
10.27.1 Description . . . 298
10.27.2 Method overview . . . 299
10.27.3 Property overview . . . 301
10.27.4 TField.Create . . . 302

10.27.5 TField.Destroy . . . 302
10.27.6 TField.Assign . . . 302
10.27.7 TField.AssignValue . . . 302
10.27.8 TField.Clear . . . 303
10.27.9 TField.FocusControl . . . 303
10.27.10TField.GetData . . . 303
10.27.11TField.IsBlob . . . 304
10.27.12TField.IsValidChar . . . 304
10.27.13TField.RefreshLookupList . . . 304
10.27.14TField.SetData . . . 304
10.27.15TField.SetFieldType . . . 305
10.27.16TField.Validate . . . 305
10.27.17TField.AsBCD . . . 305
10.27.18TField.AsBoolean . . . 306
10.27.19TField.AsCurrency . . . 306
10.27.20TField.AsDateTime . . . 306
10.27.21TField.AsFloat . . . 307
10.27.22TField.AsLongint . . . 307
10.27.23TField.AsLargeInt . . . 307
10.27.24TField.AsInteger . . . 308
10.27.25TField.AsString . . . 308
10.27.26TField.AsWideString . . . 308
10.27.27TField.AsVariant . . . 309
10.27.28TField.AttributeSet . . . 309
10.27.29TField.Calculated . . . 309
10.27.30TField.CanModify . . . 309
10.27.31TField.CurValue . . . 310
10.27.32TField.DataSet . . . 310
10.27.33TField.DataSize . . . 310
10.27.34TField.DataType . . . 310
10.27.35TField.DisplayName . . . 311
10.27.36TField.DisplayText . . . 311
10.27.37TField.FieldNo . . . 311
10.27.38TField.IsIndexField . . . 312
10.27.39TField.IsNull . . . 312
10.27.40TField.Lookup . . . 312
10.27.41TField.NewValue . . . 312
10.27.42TField.Offset . . . 313
10.27.43TField.Size . . . 313
10.27.44TField.Text . . . 313

10.27.45TField.ValidChars . . . 313
10.27.46TField.Value . . . 314
10.27.47TField.OldValue . . . 314
10.27.48TField.LookupList . . . 314
10.27.49TField.Alignment . . . 315
10.27.50TField.CustomConstraint . . . 315
10.27.51TField.ConstraintErrorMessage . . . 315
10.27.52TField.DefaultExpression . . . 316
10.27.53TField.DisplayLabel . . . 316
10.27.54TField.DisplayWidth . . . 316
10.27.55TField.FieldKind . . . 316
10.27.56TField.FieldName . . . 317
10.27.57TField.HasConstraints . . . 317
10.27.58TField.Index . . . 317
10.27.59TField.ImportedConstraint . . . 317
10.27.60TField.KeyFields . . . 318
10.27.61TField.LookupCache . . . 318
10.27.62TField.LookupDataSet . . . 318
10.27.63TField.LookupKeyFields . . . 319
10.27.64TField.LookupResultField . . . 319
10.27.65TField.Origin . . . 319
10.27.66TField.ProviderFlags . . . 319
10.27.67TField.ReadOnly . . . 320
10.27.68TField.Required . . . 320
10.27.69TField.Visible . . . 320
10.27.70TField.OnChange . . . 321
10.27.71TField.OnGetText . . . 321
10.27.72TField.OnSetText . . . 321
10.27.73TField.OnValidate . . . 322
10.28TFieldDef . . . 322
10.28.1 Description . . . 322
10.28.2 Method overview . . . 322
10.28.3 Property overview . . . 322
10.28.4 TFieldDef.Create . . . 322
10.28.5 TFieldDef.Destroy . . . 323
10.28.6 TFieldDef.Assign . . . 323
10.28.7 TFieldDef.CreateField . . . 323
10.28.8 TFieldDef.FieldClass . . . 324
10.28.9 TFieldDef.FieldNo . . . 324
10.28.10TFieldDef.InternalCalcField . . . 324

10.28.11TFieldDef.Required . . . 324
10.28.12TFieldDef.Attributes . . . 325
10.28.13TFieldDef.DataType . . . 325
10.28.14TFieldDef.Precision . . . 325
10.28.15TFieldDef.Size . . . 326
10.29TFieldDefs . . . 326
10.29.1 Description . . . 326
10.29.2 Method overview . . . 326
10.29.3 Property overview . . . 326
10.29.4 TFieldDefs.Create . . . 326
10.29.5 TFieldDefs.Add . . . 327
10.29.6 TFieldDefs.AddFieldDef . . . 327
10.29.7 TFieldDefs.Assign . . . 327
10.29.8 TFieldDefs.Find . . . 328
10.29.9 TFieldDefs.Update . . . 328
10.29.10TFieldDefs.MakeNameUnique . . . 328
10.29.11TFieldDefs.HiddenFields . . . 328
10.29.12TFieldDefs.Items . . . 329
10.30Tfields . . . 329
10.30.1 Description . . . 329
10.30.2 Method overview . . . 329
10.30.3 Property overview . . . 329
10.30.4 Tfields.Create . . . 329
10.30.5 Tfields.Destroy . . . 330
10.30.6 Tfields.Add . . . 330
10.30.7 Tfields.CheckFieldName . . . 330
10.30.8 Tfields.CheckFieldNames . . . 330
10.30.9 Tfields.Clear . . . 331
10.30.10Tfields.FindField . . . 331
10.30.11Tfields.FieldByName . . . 331
10.30.12Tfields.FieldByNumber . . . 331
10.30.13Tfields.GetEnumerator . . . 332
10.30.14Tfields.GetFieldNames . . . 332
10.30.15Tfields.IndexOf . . . 332
10.30.16Tfields.Remove . . . 332
10.30.17Tfields.Count . . . 333
10.30.18Tfields.Dataset . . . 333
10.30.19Tfields.Fields . . . 333
10.31TFieldsEnumerator . . . 333
10.31.1 Method overview . . . 333

10.31.2 Property overview . . . 334
10.31.3 TFieldsEnumerator.Create . . . 334
10.31.4 TFieldsEnumerator.MoveNext . . . 334
10.31.5 TFieldsEnumerator.Current . . . 334
10.32 TFloatField . . . 334
10.32.1 Description . . . 334
10.32.2 Method overview . . . 334
10.32.3 Property overview . . . 334
10.32.4 TFloatField.Create . . . 335
10.32.5 TFloatField.CheckRange . . . 335
10.32.6 TFloatField.Value . . . 335
10.32.7 TFloatField.Currency . . . 335
10.32.8 TFloatField.MaxValue . . . 336
10.32.9 TFloatField.MinValue . . . 336
10.32.10 TFloatField.Precision . . . 336
10.33 TGraphicField . . . 337
10.33.1 Description . . . 337
10.33.2 Method overview . . . 337
10.33.3 TGraphicField.Create . . . 337
10.34 TGuidField . . . 337
10.34.1 Description . . . 337
10.34.2 Method overview . . . 337
10.34.3 Property overview . . . 337
10.34.4 TGuidField.Create . . . 338
10.34.5 TGuidField.AsGuid . . . 338
10.35 TIndexDef . . . 338
10.35.1 Description . . . 338
10.35.2 Method overview . . . 338
10.35.3 Property overview . . . 338
10.35.4 TIndexDef.Create . . . 339
10.35.5 TIndexDef.Destroy . . . 339
10.35.6 TIndexDef.Expression . . . 339
10.35.7 TIndexDef.Fields . . . 339
10.35.8 TIndexDef.CaseInsFields . . . 340
10.35.9 TIndexDef.DescFields . . . 340
10.35.10 TIndexDef.Options . . . 340
10.35.11 TIndexDef.Source . . . 341
10.36 TIndexDefs . . . 341
10.36.1 Description . . . 341
10.36.2 Method overview . . . 341

10.36.3 Property overview . . . 341
10.36.4 TIndexDefs.Create . . . 341
10.36.5 TIndexDefs.Destroy . . . 342
10.36.6 TIndexDefs.Add . . . 342
10.36.7 TIndexDefs.AddIndexDef . . . 342
10.36.8 TIndexDefs.Find . . . 342
10.36.9 TIndexDefs.FindIndexForFields . . . 343
10.36.10TIndexDefs.GetIndexForFields . . . 343
10.36.11TIndexDefs.Update . . . 343
10.36.12TIndexDefs.Items . . . 343
10.37TLargeintField . . . 344
10.37.1 Description . . . 344
10.37.2 Method overview . . . 344
10.37.3 Property overview . . . 344
10.37.4 TLargeintField.Create . . . 344
10.37.5 TLargeintField.CheckRange . . . 344
10.37.6 TLargeintField.Value . . . 345
10.37.7 TLargeintField.MaxValue . . . 345
10.37.8 TLargeintField.MinValue . . . 345
10.38TLongintField . . . 346
10.38.1 Description . . . 346
10.38.2 Method overview . . . 346
10.38.3 Property overview . . . 346
10.38.4 TLongintField.Create . . . 346
10.38.5 TLongintField.CheckRange . . . 346
10.38.6 TLongintField.Value . . . 347
10.38.7 TLongintField.MaxValue . . . 347
10.38.8 TLongintField.MinValue . . . 347
10.39TLookupList . . . 347
10.39.1 Description . . . 347
10.39.2 Method overview . . . 348
10.39.3 TLookupList.Create . . . 348
10.39.4 TLookupList.Destroy . . . 348
10.39.5 TLookupList.Add . . . 348
10.39.6 TLookupList.Clear . . . 348
10.39.7 TLookupList.FirstKeyByValue . . . 349
10.39.8 TLookupList.ValueOfKey . . . 349
10.39.9 TLookupList.ValuesToStrings . . . 349
10.40TMasterDataLink . . . 349
10.40.1 Description . . . 349

10.40.2 Method overview . . . 350
10.40.3 Property overview . . . 350
10.40.4 TMasterDataLink.Create . . . 350
10.40.5 TMasterDataLink.Destroy . . . 350
10.40.6 TMasterDataLink.FieldNames . . . 350
10.40.7 TMasterDataLink.Fields . . . 351
10.40.8 TMasterDataLink.OnMasterChange . . . 351
10.40.9 TMasterDataLink.OnMasterDisable . . . 351
10.41 TMasterParamsDataLink . . . 351
10.41.1 Description . . . 351
10.41.2 Method overview . . . 352
10.41.3 Property overview . . . 352
10.41.4 TMasterParamsDataLink.Create . . . 352
10.41.5 TMasterParamsDataLink.RefreshParamNames . . . 352
10.41.6 TMasterParamsDataLink.CopyParamsFromMaster . . . 352
10.41.7 TMasterParamsDataLink.Params . . . 353
10.42 TMemoField . . . 353
10.42.1 Description . . . 353
10.42.2 Method overview . . . 353
10.42.3 Property overview . . . 353
10.42.4 TMemoField.Create . . . 353
10.42.5 TMemoField.Transliterate . . . 354
10.43 TNamedItem . . . 354
10.43.1 Description . . . 354
10.43.2 Property overview . . . 354
10.43.3 TNamedItem.DisplayName . . . 354
10.43.4 TNamedItem.Name . . . 354
10.44 TNumericField . . . 355
10.44.1 Description . . . 355
10.44.2 Method overview . . . 355
10.44.3 Property overview . . . 355
10.44.4 TNumericField.Create . . . 355
10.44.5 TNumericField.Alignment . . . 355
10.44.6 TNumericField.DisplayFormat . . . 356
10.44.7 TNumericField.EditFormat . . . 356
10.45 TParam . . . 356
10.45.1 Description . . . 356
10.45.2 Method overview . . . 357
10.45.3 Property overview . . . 357
10.45.4 TParam.Create . . . 357

10.45.5 TParam.Assign . . . 358
10.45.6 TParam.AssignField . . . 358
10.45.7 TParam.AssignToField . . . 358
10.45.8 TParam.AssignFieldValue . . . 359
10.45.9 TParam.AssignFromField . . . 359
10.45.10TParam.Clear . . . 359
10.45.11TParam.GetData . . . 359
10.45.12TParam.GetDataSize . . . 360
10.45.13TParam.LoadFromFile . . . 360
10.45.14TParam.LoadFromStream . . . 360
10.45.15TParam.SetBlobData . . . 360
10.45.16TParam.SetData . . . 361
10.45.17TParam.AsBlob . . . 361
10.45.18TParam.AsBoolean . . . 361
10.45.19TParam.AsCurrency . . . 361
10.45.20TParam.AsDate . . . 362
10.45.21TParam.AsDateTime . . . 362
10.45.22TParam.AsFloat . . . 362
10.45.23TParam.AsInteger . . . 362
10.45.24TParam.AsLargeInt . . . 363
10.45.25TParam.AsMemo . . . 363
10.45.26TParam.AsSmallInt . . . 363
10.45.27TParam.AsString . . . 363
10.45.28TParam.AsTime . . . 364
10.45.29TParam.AsWord . . . 364
10.45.30TParam.Bound . . . 364
10.45.31TParam.Dataset . . . 364
10.45.32TParam.IsNull . . . 365
10.45.33TParam.NativeStr . . . 365
10.45.34TParam.Text . . . 365
10.45.35TParam.Value . . . 365
10.45.36TParam.AsWideString . . . 366
10.45.37TParam.DataType . . . 366
10.45.38TParam.Name . . . 366
10.45.39TParam.NumericScale . . . 366
10.45.40TParam.ParamType . . . 367
10.45.41TParam.Precision . . . 367
10.45.42TParam.Size . . . 367
10.46TParams . . . 368
10.46.1 Description . . . 368

10.46.2 Method overview . . . 368
10.46.3 Property overview . . . 368
10.46.4 TParams.Create . . . 368
10.46.5 TParams.AddParam . . . 369
10.46.6 TParams.AssignValues . . . 369
10.46.7 TParams.CreateParam . . . 369
10.46.8 TParams.FindParam . . . 369
10.46.9 TParams.GetParamList . . . 370
10.46.10TParams.IsEqual . . . 370
10.46.11TParams.ParamByName . . . 370
10.46.12TParams.ParseSQL . . . 370
10.46.13TParams.RemoveParam . . . 371
10.46.14TParams.CopyParamValuesFromDataset . . . 371
10.46.15TParams.Dataset . . . 372
10.46.16TParams.Items . . . 372
10.46.17TParams.ParamValues . . . 372
10.47TSmallintField . . . 373
10.47.1 Description . . . 373
10.47.2 Method overview . . . 373
10.47.3 TSmallintField.Create . . . 373
10.48TStringField . . . 373
10.48.1 Description . . . 373
10.48.2 Method overview . . . 373
10.48.3 Property overview . . . 373
10.48.4 TStringField.Create . . . 374
10.48.5 TStringField.SetFieldType . . . 374
10.48.6 TStringField.FixedChar . . . 374
10.48.7 TStringField.Transliterate . . . 374
10.48.8 TStringField.Value . . . 374
10.48.9 TStringField.Size . . . 375
10.49TTimeField . . . 375
10.49.1 Description . . . 375
10.49.2 Method overview . . . 375
10.49.3 TTimeField.Create . . . 375
10.50TVarBytesField . . . 375
10.50.1 Description . . . 375
10.50.2 Method overview . . . 376
10.50.3 TVarBytesField.Create . . . 376
10.51TVariantField . . . 376
10.51.1 Description . . . 376

10.51.2 Method overview . . . 376
10.51.3 TVariantField.Create . . . 376
10.52 TWideMemoField . . . 377
10.52.1 Description . . . 377
10.52.2 Method overview . . . 377
10.52.3 Property overview . . . 377
10.52.4 TWideMemoField.Create . . . 377
10.52.5 TWideMemoField.Value . . . 377
10.53 TWideStringField . . . 378
10.53.1 Description . . . 378
10.53.2 Method overview . . . 378
10.53.3 Property overview . . . 378
10.53.4 TWideStringField.Create . . . 378
10.53.5 TWideStringField.Value . . . 378
10.54 TWordField . . . 378
10.54.1 Description . . . 378
10.54.2 Method overview . . . 379
10.54.3 TWordField.Create . . . 379

11 Reference for unit 'dbugintf' 384
11.1 Overview . . . 384
11.2 Writing a debug server . . . 384
11.3 Constants, types and variables . . . 384
11.3.1 Resource strings . . . 384
11.3.2 Constants . . . 385
11.3.3 Types . . . 385
11.4 Procedures and functions . . . 385
11.4.1 GetDebuggingEnabled . . . 385
11.4.2 InitDebugClient . . . 386
11.4.3 SendBoolean . . . 386
11.4.4 SendDateTime . . . 386
11.4.5 SendDebug . . . 386
11.4.6 SendDebugEx . . . 387
11.4.7 SendDebugFmt . . . 387
11.4.8 SendDebugFmtEx . . . 387
11.4.9 SendInteger . . . 388
11.4.10 SendMethodEnter . . . 388
11.4.11 SendMethodExit . . . 388
11.4.12 SendPointer . . . 389
11.4.13 SendSeparator . . . 389

11.4.14 SetDebuggingEnabled . . . 389
11.4.15 StartDebugServer . . . 389

12 Reference for unit 'dbugmsg' 391
12.1 Used units . . . 391
12.2 Overview . . . 391
12.3 Constants, types and variables . . . 391
12.3.1 Constants . . . 391
12.3.2 Types . . . 392
12.4 Procedures and functions . . . 392
12.4.1 DebugMessageName . . . 392
12.4.2 ReadDebugMessageFromStream . . . 392
12.4.3 WriteDebugMessageToStream . . . 393

13 Reference for unit 'eventlog' 394
13.1 Used units . . . 394
13.2 Overview . . . 394
13.3 Constants, types and variables . . . 394
13.3.1 Resource strings . . . 394
13.3.2 Types . . . 395
13.4 ELogError . . . 395
13.4.1 Description . . . 395
13.5 TEventLog . . . 395
13.5.1 Description . . . 395
13.5.2 Method overview . . . 396
13.5.3 Property overview . . . 396
13.5.4 TEventLog.Destroy . . . 396
13.5.5 TEventLog.EventTypeToString . . . 396
13.5.6 TEventLog.RegisterMessageFile . . . 397
13.5.7 TEventLog.Log . . . 398
13.5.8 TEventLog.Warning . . . 398
13.5.9 TEventLog.Error . . . 398
13.5.10 TEventLog.Debug . . . 399
13.5.11 TEventLog.Info . . . 399
13.5.12 TEventLog.Identification . . . 399
13.5.13 TEventLog.LogType . . . 399
13.5.14 TEventLog.Active . . . 400
13.5.15 TEventLog.RaiseExceptionOnError . . . 400
13.5.16 TEventLog.DefaultEventType . . . 400
13.5.17 TEventLog.FileName . . . 400
13.5.18 TEventLog.TimeStampFormat . . . 401

13.5.19 TEventLog.CustomLogType . . . 401
13.5.20 TEventLog.EventIDOffset . . . 401
13.5.21 TEventLog.OnGetCustomCategory . . . 402
13.5.22 TEventLog.OnGetCustomEventID . . . 402
13.5.23 TEventLog.OnGetCustomEvent . . . 402

14 Reference for unit 'ezcgi' 403
14.1 Used units . . . 403
14.2 Overview . . . 403
14.3 Constants, types and variables . . . 403
14.3.1 Constants . . . 403
14.4 ECGIException . . . 403
14.4.1 Description . . . 403
14.5 TEZcgi . . . 404
14.5.1 Description . . . 404
14.5.2 Method overview . . . 404
14.5.3 Property overview . . . 404
14.5.4 TEZcgi.Create . . . 404
14.5.5 TEZcgi.Destroy . . . 404
14.5.6 TEZcgi.Run . . . 405
14.5.7 TEZcgi.WriteContent . . . 405
14.5.8 TEZcgi.PutLine . . . 405
14.5.9 TEZcgi.GetValue . . . 406
14.5.10 TEZcgi.DoPost . . . 406
14.5.11 TEZcgi.DoGet . . . 406
14.5.12 TEZcgi.Values . . . 406
14.5.13 TEZcgi.Names . . . 407
14.5.14 TEZcgi.Variables . . . 407
14.5.15 TEZcgi.VariableCount . . . 408
14.5.16 TEZcgi.Name . . . 408
14.5.17 TEZcgi.Email . . . 408

15 Reference for unit 'fpTimer' 409
15.1 Used units . . . 409
15.2 Overview . . . 409
15.3 Constants, types and variables . . . 409
15.3.1 Types . . . 409
15.3.2 Variables . . . 409
15.4 TFPCustomTimer . . . 410
15.4.1 Description . . . 410
15.4.2 Method overview . . . 410

15.4.3 TFPCustomTimer.Create . . . 410
15.4.4 TFPCustomTimer.Destroy . . . 410
15.4.5 TFPCustomTimer.StartTimer . . . 411
15.4.6 TFPCustomTimer.StopTimer . . . 411
15.5 TFPTimer . . . 411
15.5.1 Description . . . 411
15.5.2 Property overview . . . 411
15.5.3 TFPTimer.Enabled . . . 411
15.5.4 TFPTimer.Interval . . . 412
15.5.5 TFPTimer.OnTimer . . . 412
15.6 TFPTimerDriver . . . 412
15.6.1 Description . . . 412
15.6.2 Method overview . . . 412
15.6.3 Property overview . . . 412
15.6.4 TFPTimerDriver.Create . . . 413
15.6.5 TFPTimerDriver.StartTimer . . . 413
15.6.6 TFPTimerDriver.StopTimer . . . 413
15.6.7 TFPTimerDriver.Timer . . . 413

16 Reference for unit 'gettext' 414
16.1 Used units . . . 414
16.2 Overview . . . 414
16.3 Constants, types and variables . . . 414
16.3.1 Constants . . . 414
16.3.2 Types . . . 414
16.4 Procedures and functions . . . 415
16.4.1 GetLanguageIDs . . . 415
16.4.2 TranslateResourceStrings . . . 416
16.4.3 TranslateUnitResourceStrings . . . 416
16.5 EMOFileError . . . 416
16.5.1 Description . . . 416
16.6 TMOFile . . . 416
16.6.1 Description . . . 416
16.6.2 Method overview . . . 417
16.6.3 TMOFile.Create . . . 417
16.6.4 TMOFile.Destroy . . . 417
16.6.5 TMOFile.Translate . . . 417

17 Reference for unit 'idea' 418
17.1 Used units . . . 418
17.2 Overview . . . 418

17.3 Constants, types and variables . . . 418
17.3.1 Constants . . . 418
17.3.2 Types . . . 419
17.4 Procedures and functions . . . 419
17.4.1 CipherIdea . . . 419
17.4.2 DeKeyIdea . . . 419
17.4.3 EnKeyIdea . . . 420
17.5 EIDEAError . . . 420
17.5.1 Description . . . 420
17.6 TIDEADeCryptStream . . . 420
17.6.1 Description . . . 420
17.6.2 Method overview . . . 420
17.6.3 TIDEADeCryptStream.Create . . . 420
17.6.4 TIDEADeCryptStream.Read . . . 421
17.6.5 TIDEADeCryptStream.Seek . . . 421
17.7 TIDEAEncryptStream . . . 421
17.7.1 Description . . . 421
17.7.2 Method overview . . . 422
17.7.3 TIDEAEncryptStream.Create . . . 422
17.7.4 TIDEAEncryptStream.Destroy . . . 422
17.7.5 TIDEAEncryptStream.Write . . . 422
17.7.6 TIDEAEncryptStream.Seek . . . 423
17.7.7 TIDEAEncryptStream.Flush . . . 423
17.8 TIDEAStream . . . 423
17.8.1 Description . . . 423
17.8.2 Method overview . . . 423
17.8.3 Property overview . . . 423
17.8.4 TIDEAStream.Create . . . 424
17.8.5 TIDEAStream.Key . . . 424

18 Reference for unit 'inicol' 425
18.1 Used units . . . 425
18.2 Overview . . . 425
18.3 Constants, types and variables . . . 425
18.3.1 Constants . . . 425
18.4 EIniCol . . . 426
18.4.1 Description . . . 426
18.5 TIniCollection . . . 426
18.5.1 Description . . . 426
18.5.2 Method overview . . . 426

18.5.3 Property overview . . . 426
18.5.4 TIniCollection.Load . . . 426
18.5.5 TIniCollection.Save . . . 427
18.5.6 TIniCollection.SaveToIni . . . 427
18.5.7 TIniCollection.SaveToFile . . . 427
18.5.8 TIniCollection.LoadFromIni . . . 428
18.5.9 TIniCollection.LoadFromFile . . . 428
18.5.10 TIniCollection.Prefix . . . 428
18.5.11 TIniCollection.SectionPrefix . . . 429
18.5.12 TIniCollection.FileName . . . 429
18.5.13 TIniCollection.GlobalSection . . . 429
18.6 TIniCollectionItem . . . 429
18.6.1 Description . . . 429
18.6.2 Method overview . . . 430
18.6.3 Property overview . . . 430
18.6.4 TIniCollectionItem.SaveToIni . . . 430
18.6.5 TIniCollectionItem.LoadFromIni . . . 430
18.6.6 TIniCollectionItem.SaveToFile . . . 430
18.6.7 TIniCollectionItem.LoadFromFile . . . 431
18.6.8 TIniCollectionItem.SectionName . . . 431
18.7 TNamedIniCollection . . . 431
18.7.1 Description . . . 431
18.7.2 Method overview . . . 431
18.7.3 Property overview . . . 432
18.7.4 TNamedIniCollection.IndexOfUserData . . . 432
18.7.5 TNamedIniCollection.IndexOfName . . . 432
18.7.6 TNamedIniCollection.FindByName . . . 432
18.7.7 TNamedIniCollection.FindByUserData . . . 433
18.7.8 TNamedIniCollection.NamedItems . . . 433
18.8 TNamedIniCollectionItem . . . 433
18.8.1 Description . . . 433
18.8.2 Property overview . . . 433
18.8.3 TNamedIniCollectionItem.UserData . . . 433
18.8.4 TNamedIniCollectionItem.Name . . . 434

19 Reference for unit 'IniFiles' 435
19.1 Used units . . . 435
19.2 Overview . . . 435
19.3 TCustomIniFile . . . 435
19.3.1 Description . . . 435

19.3.2 Method overview . . . 436
19.3.3 Property overview . . . 436
19.3.4 TCustomIniFile.Create . . . 436
19.3.5 TCustomIniFile.Destroy . . . 437
19.3.6 TCustomIniFile.SectionExists . . . 437
19.3.7 TCustomIniFile.ReadString . . . 437
19.3.8 TCustomIniFile.WriteString . . . 438
19.3.9 TCustomIniFile.ReadInteger . . . 438
19.3.10 TCustomIniFile.WriteInteger . . . 438
19.3.11 TCustomIniFile.ReadBool . . . 438
19.3.12 TCustomIniFile.WriteBool . . . 439
19.3.13 TCustomIniFile.ReadDate . . . 439
19.3.14 TCustomIniFile.ReadDateTime . . . 439
19.3.15 TCustomIniFile.ReadFloat . . . 440
19.3.16 TCustomIniFile.ReadTime . . . 440
19.3.17 TCustomIniFile.ReadBinaryStream . . . 440
19.3.18 TCustomIniFile.WriteDate . . . 441
19.3.19 TCustomIniFile.WriteDateTime . . . 441
19.3.20 TCustomIniFile.WriteFloat . . . 441
19.3.21 TCustomIniFile.WriteTime . . . 442
19.3.22 TCustomIniFile.WriteBinaryStream . . . 442
19.3.23 TCustomIniFile.ReadSection . . . 442
19.3.24 TCustomIniFile.ReadSections . . . 443
19.3.25 TCustomIniFile.ReadSectionValues . . . 443
19.3.26 TCustomIniFile.EraseSection . . . 443
19.3.27 TCustomIniFile.DeleteKey . . . 443
19.3.28 TCustomIniFile.UpdateFile . . . 444
19.3.29 TCustomIniFile.ValueExists . . . 444
19.3.30 TCustomIniFile.FileName . . . 444
19.3.31 TCustomIniFile.EscapeLineFeeds . . . 445
19.3.32 TCustomIniFile.CaseSensitive . . . 445
19.3.33 TCustomIniFile.StripQuotes . . . 445
19.4 THashedStringList . . . 445
19.4.1 Description . . . 445
19.4.2 Method overview . . . 446
19.4.3 THashedStringList.Create . . . 446
19.4.4 THashedStringList.Destroy . . . 446
19.4.5 THashedStringList.IndexOf . . . 446
19.4.6 THashedStringList.IndexOfName . . . 446
19.5 TIniFile . . . 447

19.5.1 Description . . . 447
19.5.2 Method overview . . . 447
19.5.3 Property overview . . . 447
19.5.4 TIniFile.Create . . . 447
19.5.5 TIniFile.Destroy . . . 447
19.5.6 TIniFile.ReadString . . . 448
19.5.7 TIniFile.WriteString . . . 448
19.5.8 TIniFile.ReadSection . . . 448
19.5.9 TIniFile.ReadSectionRaw . . . 448
19.5.10 TIniFile.ReadSections . . . 449
19.5.11 TIniFile.ReadSectionValues . . . 449
19.5.12 TIniFile.EraseSection . . . 449
19.5.13 TIniFile.DeleteKey . . . 449
19.5.14 TIniFile.UpdateFile . . . 450
19.5.15 TIniFile.Stream . . . 450
19.5.16 TIniFile.CacheUpdates . . . 450
19.6 TIniFileKey . . . 451
19.6.1 Description . . . 451
19.6.2 Method overview . . . 451
19.6.3 Property overview . . . 451
19.6.4 TIniFileKey.Create . . . 451
19.6.5 TIniFileKey.Ident . . . 451
19.6.6 TIniFileKey.Value . . . 451
19.7 TIniFileKeyList . . . 452
19.7.1 Description . . . 452
19.7.2 Method overview . . . 452
19.7.3 Property overview . . . 452
19.7.4 TIniFileKeyList.Destroy . . . 452
19.7.5 TIniFileKeyList.Clear . . . 452
19.7.6 TIniFileKeyList.Items . . . 452
19.8 TIniFileSection . . . 453
19.8.1 Description . . . 453
19.8.2 Method overview . . . 453
19.8.3 Property overview . . . 453
19.8.4 TIniFileSection.Empty . . . 453
19.8.5 TIniFileSection.Create . . . 453
19.8.6 TIniFileSection.Destroy . . . 453
19.8.7 TIniFileSection.Name . . . 454
19.8.8 TIniFileSection.KeyList . . . 454
19.9 TIniFileSectionList . . . 454

19.9.1 Description . . . 454
19.9.2 Method overview . . . 454
19.9.3 Property overview . . . 454
19.9.4 TIniFileSectionList.Destroy . . . 455
19.9.5 TIniFileSectionList.Clear . . . 455
19.9.6 TIniFileSectionList.Items . . . 455
19.10TMemIniFile . . . 455
19.10.1 Description . . . 455
19.10.2 Method overview . . . 455
19.10.3 TMemIniFile.Create . . . 456
19.10.4 TMemIniFile.Clear . . . 456
19.10.5 TMemIniFile.GetStrings . . . 456
19.10.6 TMemIniFile.Rename . . . 456
19.10.7 TMemIniFile.SetStrings . . . 457

20 Reference for unit 'iostream' 458
20.1 Used units . . . 458
20.2 Overview . . . 458
20.3 Constants, types and variables . . . 458
20.3.1 Types . . . 458
20.4 EIOStreamError . . . 459
20.4.1 Description . . . 459
20.5 TIOStream . . . 459
20.5.1 Description . . . 459
20.5.2 Method overview . . . 459
20.5.3 TIOStream.Create . . . 459
20.5.4 TIOStream.Read . . . 459
20.5.5 TIOStream.Write . . . 460
20.5.6 TIOStream.SetSize . . . 460
20.5.7 TIOStream.Seek . . . 460

21 Reference for unit 'libtar' 461
21.1 Used units . . . 461
21.2 Overview . . . 461
21.3 Constants, types and variables . . . 461
21.3.1 Constants . . . 461
21.3.2 Types . . . 462
21.4 Procedures and functions . . . 464
21.4.1 ClearDirRec . . . 464
21.4.2 ConvertFilename . . . 464
21.4.3 FileTimeGMT . . . 464

21.4.4 PermissionString . . . 464
21.5 TTarArchive . . . 465
21.5.1 Description . . . 465
21.5.2 Method overview . . . 465
21.5.3 TTarArchive.Create . . . 465
21.5.4 TTarArchive.Destroy . . . 465
21.5.5 TTarArchive.Reset . . . 465
21.5.6 TTarArchive.FindNext . . . 466
21.5.7 TTarArchive.ReadFile . . . 466
21.5.8 TTarArchive.GetFilePos . . . 466
21.5.9 TTarArchive.SetFilePos . . . 467
21.6 TTarWriter . . . 467
21.6.1 Description . . . 467
21.6.2 Method overview . . . 467
21.6.3 Property overview . . . 467
21.6.4 TTarWriter.Create . . . 467
21.6.5 TTarWriter.Destroy . . . 468
21.6.6 TTarWriter.AddFile . . . 468
21.6.7 TTarWriter.AddStream . . . 468
21.6.8 TTarWriter.AddString . . . 469
21.6.9 TTarWriter.AddDir . . . 469
21.6.10 TTarWriter.AddSymbolicLink . . . 469
21.6.11 TTarWriter.AddLink . . . 470
21.6.12 TTarWriter.AddVolumeHeader . . . 470
21.6.13 TTarWriter.Finalize . . . 470
21.6.14 TTarWriter.Permissions . . . 470
21.6.15 TTarWriter.UID . . . 471
21.6.16 TTarWriter.GID . . . 471
21.6.17 TTarWriter.UserName . . . 471
21.6.18 TTarWriter.GroupName . . . 471
21.6.19 TTarWriter.Mode . . . 472
21.6.20 TTarWriter.Magic . . . 472

22 Reference for unit 'Pipes' 473
22.1 Used units . . . 473
22.2 Overview . . . 473
22.3 Constants, types and variables . . . 473
22.3.1 Constants . . . 473
22.4 Procedures and functions . . . 473
22.4.1 CreatePipeHandles . . . 473

22.4.2 CreatePipeStreams . . . 474
22.5 EPipeCreation . . . 474
22.5.1 Description . . . 474
22.6 EPipeError . . . 474
22.6.1 Description . . . 474
22.7 EPipeSeek . . . 474
22.7.1 Description . . . 474
22.8 TInputPipeStream . . . 474
22.8.1 Description . . . 474
22.8.2 Method overview . . . 475
22.8.3 Property overview . . . 475
22.8.4 TInputPipeStream.Write . . . 475
22.8.5 TInputPipeStream.Seek . . . 475
22.8.6 TInputPipeStream.Read . . . 476
22.8.7 TInputPipeStream.NumBytesAvailable . . . 476
22.9 TOutputPipeStream . . . 476
22.9.1 Description . . . 476
22.9.2 Method overview . . . 476
22.9.3 TOutputPipeStream.Seek . . . 476
22.9.4 TOutputPipeStream.Read . . . 477

23 Reference for unit 'pooledmm' 478
23.1 Used units . . . 478
23.2 Overview . . . 478
23.3 Constants, types and variables . . . 478
23.3.1 Types . . . 478
23.4 TNonFreePooledMemManager . . . 479
23.4.1 Description . . . 479
23.4.2 Method overview . . . 479
23.4.3 Property overview . . . 479
23.4.4 TNonFreePooledMemManager.Clear . . . 479
23.4.5 TNonFreePooledMemManager.Create . . . 479
23.4.6 TNonFreePooledMemManager.Destroy . . . 480
23.4.7 TNonFreePooledMemManager.NewItem . . . 480
23.4.8 TNonFreePooledMemManager.EnumerateItems . . . 480
23.4.9 TNonFreePooledMemManager.ItemSize . . . 480
23.5 TPooledMemManager . . . 481
23.5.1 Description . . . 481
23.5.2 Method overview . . . 481
23.5.3 Property overview . . . 481

23.5.4 TPooledMemManager.Clear . . . 481
23.5.5 TPooledMemManager.Create . . . 481
23.5.6 TPooledMemManager.Destroy . . . 481
23.5.7 TPooledMemManager.MinimumFreeCount . . . 482
23.5.8 TPooledMemManager.MaximumFreeCountRatio . . . 482
23.5.9 TPooledMemManager.Count . . . 482
23.5.10 TPooledMemManager.FreeCount . . . 483
23.5.11 TPooledMemManager.AllocatedCount . . . 483
23.5.12 TPooledMemManager.FreedCount . . . 483

24 Reference for unit 'process' 484
24.1 Used units . . . 484
24.2 Overview . . . 484
24.3 Constants, types and variables . . . 484
24.3.1 Types . . . 484
24.4 EProcess . . . 486
24.4.1 Description . . . 486
24.5 TProcess . . . 486
24.5.1 Description . . . 486
24.5.2 Method overview . . . 487
24.5.3 Property overview . . . 487
24.5.4 TProcess.Create . . . 488
24.5.5 TProcess.Destroy . . . 488
24.5.6 TProcess.Execute . . . 488
24.5.7 TProcess.CloseInput . . . 489
24.5.8 TProcess.CloseOutput . . . 489
24.5.9 TProcess.CloseStderr . . . 489
24.5.10 TProcess.Resume . . . 489
24.5.11 TProcess.Suspend . . . 490
24.5.12 TProcess.Terminate . . . 490
24.5.13 TProcess.WaitOnExit . . . 490
24.5.14 TProcess.WindowRect . . . 491
24.5.15 TProcess.Handle . . . 491
24.5.16 TProcess.ProcessHandle . . . 491
24.5.17 TProcess.ThreadHandle . . . 491
24.5.18 TProcess.ProcessID . . . 492
24.5.19 TProcess.ThreadID . . . 492
24.5.20 TProcess.Input . . . 492
24.5.21 TProcess.Output . . . 493
24.5.22 TProcess.Stderr . . . 493

24.5.23 TProcess.ExitStatus . . . 493
24.5.24 TProcess.InheritHandles . . . 494
24.5.25 TProcess.Active . . . 494
24.5.26 TProcess.ApplicationName . . . 494
24.5.27 TProcess.CommandLine . . . 494
24.5.28 TProcess.ConsoleTitle . . . 495
24.5.29 TProcess.CurrentDirectory . . . 495
24.5.30 TProcess.Desktop . . . 495
24.5.31 TProcess.Environment . . . 496
24.5.32 TProcess.Options . . . 496
24.5.33 TProcess.Priority . . . 497
24.5.34 TProcess.StartupOptions . . . 497
24.5.35 TProcess.Running . . . 498
24.5.36 TProcess.ShowWindow . . . 498
24.5.37 TProcess.WindowColumns . . . 499
24.5.38 TProcess.WindowHeight . . . 499
24.5.39 TProcess.WindowLeft . . . 499
24.5.40 TProcess.WindowRows . . . 500
24.5.41 TProcess.WindowTop . . . 500
24.5.42 TProcess.WindowWidth . . . 500
24.5.43 TProcess.FillAttribute . . . 501

25 Reference for unit 'rttiutils' 502
25.1 Used units . . . 502
25.2 Overview . . . 502
25.3 Constants, types and variables . . . 502
25.3.1 Constants . . . 502
25.3.2 Types . . . 502
25.3.3 Variables . . . 503
25.4 Procedures and functions . . . 503
25.4.1 CreateStoredItem . . . 503
25.4.2 ParseStoredItem . . . 504
25.4.3 UpdateStoredList . . . 504
25.5 TPropInfoList . . . 504
25.5.1 Description . . . 504
25.5.2 Method overview . . . 504
25.5.3 Property overview . . . 505
25.5.4 TPropInfoList.Create . . . 505
25.5.5 TPropInfoList.Destroy . . . 505
25.5.6 TPropInfoList.Contains . . . 505

25.5.7 TPropInfoList.Find . . . 505
25.5.8 TPropInfoList.Delete . . . 506
25.5.9 TPropInfoList.Intersect . . . 506
25.5.10 TPropInfoList.Count . . . 506
25.5.11 TPropInfoList.Items . . . 506
25.6 TPropsStorage . . . 507
25.6.1 Description . . . 507
25.6.2 Method overview . . . 507
25.6.3 Property overview . . . 507
25.6.4 TPropsStorage.StoreAnyProperty . . . 507
25.6.5 TPropsStorage.LoadAnyProperty . . . 507
25.6.6 TPropsStorage.StoreProperties . . . 508
25.6.7 TPropsStorage.LoadProperties . . . 508
25.6.8 TPropsStorage.LoadObjectsProps . . . 508
25.6.9 TPropsStorage.StoreObjectsProps . . . 509
25.6.10 TPropsStorage.AObject . . . 510
25.6.11 TPropsStorage.Prefix . . . 510
25.6.12 TPropsStorage.Section . . . 510
25.6.13 TPropsStorage.OnReadString . . . 510
25.6.14 TPropsStorage.OnWriteString . . . 511
25.6.15 TPropsStorage.OnEraseSection . . . 511

26 Reference for unit 'simpleipc' 512
26.1 Used units . . . 512
26.2 Overview . . . 512
26.3 Constants, types and variables . . . 512
26.3.1 Resource strings . . . 512
26.3.2 Constants . . . 513
26.3.3 Types . . . 513
26.3.4 Variables . . . 513
26.4 EIPCError . . . 514
26.4.1 Description . . . 514
26.5 TIPCClientComm . . . 514
26.5.1 Description . . . 514
26.5.2 Method overview . . . 514
26.5.3 Property overview . . . 514
26.5.4 TIPCClientComm.Create . . . 514
26.5.5 TIPCClientComm.Connect . . . 514
26.5.6 TIPCClientComm.Disconnect . . . 515
26.5.7 TIPCClientComm.ServerRunning . . . 515

26.5.8 TIPCClientComm.SendMessage . . . 515
26.5.9 TIPCClientComm.Owner . . . 516
26.6 TIPCServerComm . . . 516
26.6.1 Description . . . 516
26.6.2 Method overview . . . 516
26.6.3 Property overview . . . 516
26.6.4 TIPCServerComm.Create . . . 516
26.6.5 TIPCServerComm.StartServer . . . 517
26.6.6 TIPCServerComm.StopServer . . . 517
26.6.7 TIPCServerComm.PeekMessage . . . 517
26.6.8 TIPCServerComm.ReadMessage . . . 518
26.6.9 TIPCServerComm.Owner . . . 518
26.6.10 TIPCServerComm.InstanceID . . . 518
26.7 TSimpleIPC . . . 518
26.7.1 Description . . . 518
26.7.2 Property overview . . . 518
26.7.3 TSimpleIPC.Active . . . 519
26.7.4 TSimpleIPC.ServerID . . . 519
26.8 TSimpleIPCClient . . . 519
26.8.1 Description . . . 519
26.8.2 Method overview . . . 519
26.8.3 Property overview . . . 520
26.8.4 TSimpleIPCClient.Create . . . 520
26.8.5 TSimpleIPCClient.Destroy . . . 520
26.8.6 TSimpleIPCClient.Connect . . . 520
26.8.7 TSimpleIPCClient.Disconnect . . . 521
26.8.8 TSimpleIPCClient.ServerRunning . . . 521
26.8.9 TSimpleIPCClient.SendMessage . . . 521
26.8.10 TSimpleIPCClient.SendStringMessage . . . 521
26.8.11 TSimpleIPCClient.SendStringMessageFmt . . . 522
26.8.12 TSimpleIPCClient.ServerInstance . . . 522
26.9 TSimpleIPCServer . . . 522
26.9.1 Description . . . 522
26.9.2 Method overview . . . 523
26.9.3 Property overview . . . 523
26.9.4 TSimpleIPCServer.Create . . . 523
26.9.5 TSimpleIPCServer.Destroy . . . 523
26.9.6 TSimpleIPCServer.StartServer . . . 523
26.9.7 TSimpleIPCServer.StopServer . . . 524
26.9.8 TSimpleIPCServer.PeekMessage . . . 524

26.9.9 TSimpleIPCServer.GetMessageData . . . 524
26.9.10 TSimpleIPCServer.StringMessage . . . 525
26.9.11 TSimpleIPCServer.MsgType . . . 525
26.9.12 TSimpleIPCServer.MsgData . . . 525
26.9.13 TSimpleIPCServer.InstanceID . . . 525
26.9.14 TSimpleIPCServer.Global . . . 526
26.9.15 TSimpleIPCServer.OnMessage . . . 526

27 Reference for unit 'streamcoll' 527
27.1 Used units . . . 527
27.2 Overview . . . 527
27.3 Procedures and functions . . . 527
27.3.1 ColReadBoolean . . . 527
27.3.2 ColReadCurrency . . . 528
27.3.3 ColReadDateTime . . . 528
27.3.4 ColReadFloat . . . 528
27.3.5 ColReadInteger . . . 528
27.3.6 ColReadString . . . 529
27.3.7 ColWriteBoolean . . . 529
27.3.8 ColWriteCurrency . . . 529
27.3.9 ColWriteDateTime . . . 529
27.3.10 ColWriteFloat . . . 530
27.3.11 ColWriteInteger . . . 530
27.3.12 ColWriteString . . . 530
27.4 EStreamColl . . . 530
27.4.1 Description . . . 530
27.5 TStreamCollection . . . 530
27.5.1 Description . . . 530
27.5.2 Method overview . . . 531
27.5.3 Property overview . . . 531
27.5.4 TStreamCollection.LoadFromStream . . . 531
27.5.5 TStreamCollection.SaveToStream . . . 531
27.5.6 TStreamCollection.Streaming . . . 531
27.6 TStreamCollectionItem . . . 532
27.6.1 Description . . . 532

28 Reference for unit 'streamex' 533
28.1 Used units . . . 533
28.2 Overview . . . 533
28.3 TBidirBinaryObjectReader . . . 533
28.3.1 Description . . . 533

28.3.2 Property overview . . . 533
28.3.3 TBidirBinaryObjectReader.Position . . . 533
28.4 TBidirBinaryObjectWriter . . . 534
28.4.1 Description . . . 534
28.4.2 Property overview . . . 534
28.4.3 TBidirBinaryObjectWriter.Position . . . 534
28.5 TDelphiReader . . . 534
28.5.1 Description . . . 534
28.5.2 Method overview . . . 534
28.5.3 Property overview . . . 534
28.5.4 TDelphiReader.GetDriver . . . 535
28.5.5 TDelphiReader.ReadStr . . . 535
28.5.6 TDelphiReader.Read . . . 535
28.5.7 TDelphiReader.Position . . . 535
28.6 TDelphiWriter . . . 535
28.6.1 Description . . . 535
28.6.2 Method overview . . . 536
28.6.3 Property overview . . . 536
28.6.4 TDelphiWriter.GetDriver . . . 536
28.6.5 TDelphiWriter.FlushBuffer . . . 536
28.6.6 TDelphiWriter.Write . . . 536
28.6.7 TDelphiWriter.WriteStr . . . 536
28.6.8 TDelphiWriter.WriteValue . . . 537
28.6.9 TDelphiWriter.Position . . . 537

29 Reference for unit 'StreamIO' 538
29.1 Used units . . . 538
29.2 Overview . . . 538
29.3 Procedures and functions . . . 538
29.3.1 AssignStream . . . 538
29.3.2 GetStream . . . 539

30 Reference for unit 'syncobjs' 540
30.1 Used units . . . 540
30.2 Overview . . . 540
30.3 Constants, types and variables . . . 540
30.3.1 Constants . . . 540
30.3.2 Types . . . 540
30.4 TCriticalSection . . . 541
30.4.1 Description . . . 541
30.4.2 Method overview . . . 541

30.4.3 TCriticalSection.Acquire . . . 542
30.4.4 TCriticalSection.Release . . . 542
30.4.5 TCriticalSection.Enter . . . 542
30.4.6 TCriticalSection.TryEnter . . . 542
30.4.7 TCriticalSection.Leave . . . 543
30.4.8 TCriticalSection.Create . . . 543
30.4.9 TCriticalSection.Destroy . . . 543
30.5 TEventObject . . . 543
30.5.1 Description . . . 543
30.5.2 Method overview . . . 544
30.5.3 Property overview . . . 544
30.5.4 TEventObject.Create . . . 544
30.5.5 TEventObject.destroy . . . 544
30.5.6 TEventObject.ResetEvent . . . 544
30.5.7 TEventObject.SetEvent . . . 545
30.5.8 TEventObject.WaitFor . . . 545
30.5.9 TEventObject.ManualReset . . . 545
30.6 THandleObject . . . 545
30.6.1 Description . . . 545
30.6.2 Method overview . . . 545
30.6.3 Property overview . . . 546
30.6.4 THandleObject.destroy . . . 546
30.6.5 THandleObject.Handle . . . 546
30.6.6 THandleObject.LastError . . . 546
30.7 TSimpleEvent . . . 546
30.7.1 Description . . . 546
30.7.2 Method overview . . . 546
30.7.3 TSimpleEvent.Create . . . 547
30.8 TSynchroObject . . . 547
30.8.1 Description . . . 547
30.8.2 Method overview . . . 547
30.8.3 TSynchroObject.Acquire . . . 547
30.8.4 TSynchroObject.Release . . . 547

31 Reference for unit 'URIParser' 548
31.1 Overview . . . 548
31.2 Constants, types and variables . . . 548
31.2.1 Types . . . 548
31.3 Procedures and functions . . . 548
31.3.1 EncodeURI . . . 548

31.3.2 FilenameToURI . . . 549
31.3.3 IsAbsoluteURI . . . 549
31.3.4 ParseURI . . . 549
31.3.5 ResolveRelativeURI . . . 550
31.3.6 URIToFilename . . . 550

32 Reference for unit 'zstream' 551
32.1 Used units . . . 551
32.2 Overview . . . 551
32.3 Constants, types and variables . . . 551
32.3.1 Types . . . 551
32.4 Ecompressionerror . . . 552
32.4.1 Description . . . 552
32.5 Edecompressionerror . . . 552
32.5.1 Description . . . 552
32.6 Egzfileerror . . . 552
32.6.1 Description . . . 552
32.7 Ezliberror . . . 552
32.7.1 Description . . . 552
32.8 Tcompressionstream . . . 552
32.8.1 Description . . . 552
32.8.2 Method overview . . . 553
32.8.3 Tcompressionstream.create . . . 553
32.8.4 Tcompressionstream.destroy . . . 553
32.8.5 Tcompressionstream.write . . . 553
32.8.6 Tcompressionstream.flush . . . 554
32.8.7 Tcompressionstream.get_compressionrate . . . 554
32.9 Tcustomzlibstream . . . 554
32.9.1 Description . . . 554
32.9.2 Method overview . . . 554
32.9.3 Tcustomzlibstream.create . . . 554
32.9.4 Tcustomzlibstream.destroy . . . 555
32.10 Tdecompressionstream . . . 555
32.10.1 Description . . . 555
32.10.2 Method overview . . . 555
32.10.3 Tdecompressionstream.create . . . 555
32.10.4 Tdecompressionstream.destroy . . . 555
32.10.5 Tdecompressionstream.read . . . 556
32.10.6 Tdecompressionstream.seek . . . 556
32.10.7 Tdecompressionstream.get_compressionrate . . . 557

32.11TGZFileStream 557
32.11.1 Description 557
32.11.2 Method overview 557
32.11.3 TGZFileStream.create 557
32.11.4 TGZFileStream.read 557
32.11.5 TGZFileStream.write 558
32.11.6 TGZFileStream.seek 558
32.11.7 TGZFileStream.destroy 558

About this guide

This document describes all constants, types, variables, functions and procedures as they are declared in the units that come standard with the FCL (Free Component Library).

Throughout this document, we will refer to functions, types and variables with `typewriter` font. Functions and procedures gave their own subsections, and for each function or procedure we have the following topics:

Declaration The exact declaration of the function.

Description What does the procedure exactly do ?

Errors What errors can occur.

See Also Cross references to other related functions/commands.

0.1 Overview

The Free Component Library is a series of units that implemenent 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	**??**
sysutils	**??**

1.2 Overview

The ascii85 provides an ASCII 85 or base 85 decoding algorithm. It is class and stream based: the TASCII85DecoderStream (55) 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 (418) or base64 (74) units.

1.3 Constants, types and variables

1.3.1 Types

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

`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.

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.

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
57	Close	Close decoder
57	ClosedP	Check if the state is correct
56	Create	Create new ASCII 85 decoder stream
56	Decode	Decode source byte
57	Destroy	Clean up instance
57	Read	Read data from stream
58	Seek	Set stream position

1.4.3 Property overview

Page	Property	Access	Description
58	BExpectBoundary	rw	Expect čharacter

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 (57)

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 (57)

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 (57), TASCII85DecoderStream.Read (57), TASCII85DecoderStream.Destroy (57)

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 (57), TASCII85DecoderStream.BExpectBoundary (58)

1.4.8 TASCII85DecoderStream.Destroy

Synopsis: Clean up instance

Declaration: `destructor Destroy;  Override`

Visibility: public

Description: `Destroy` closes the input stream using Close (57) and cleans up the `TASCII85DecoderStream` instance from memory.

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

See also: TASCII85DecoderStream.Close (57)

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 (57)

1.4.11 TASCII85DecoderStream.BExpectBoundary

Synopsis: Expect c̃haracter

Declaration:
```
Property BExpectBoundary : Boolean
```

Visibility: published

Access: Read,Write

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

See also: TASCII85DecoderStream.ClosedP (57)

1.5 TASCII85EncoderStream

1.5.1 Method overview

Page	Property	Description
58	Create	
59	Destroy	
59	Write	

1.5.2 Property overview

Page	Property	Access	Description
59	Boundary	r	
59	Width	r	

1.5.3 TASCII85EncoderStream.Create

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

Visibility: public

1.5.4 TASCII85EncoderStream.Destroy

Declaration: `destructor Destroy;  Override`

Visibility: public

1.5.5 TASCII85EncoderStream.Write

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

Visibility: public

1.5.6 TASCII85EncoderStream.Width

Declaration: `Property Width : Integer`

Visibility: public

Access: Read

1.5.7 TASCII85EncoderStream.Boundary

Declaration: `Property Boundary : Boolean`

Visibility: public

Access: Read

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

1.6.2 Method overview

Page	Property	Description
60	Read	Read data from the internal buffer
59	Write	Write data to the internal buffer

1.6.3 Property overview

Page	Property	Access	Description
60	FillCount	r	Number of bytes in buffer
60	Size	r	Size of buffer

1.6.4 TASCII85RingBuffer.Write

Synopsis: Write data to the internal buffer

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

Visibility: published

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

See also: TASCII85RingBuffer.FillCount (60), TASCII85RingBuffer.Read (60), TASCII85RingBuffer.Size (60)

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 (60), TASCII85RingBuffer.Write (59), TASCII85RingBuffer.Size (60)

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 (59), TASCII85RingBuffer.Read (60), TASCII85RingBuffer.Size (60)

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 (60)

Chapter 2

Reference for unit 'AVL_Tree'

2.1 Used units

Table 2.1: Used units by unit 'AVL_Tree'

Name	Page
Classes	??
sysutils	??

2.2 Overview

The `avl_tree` unit implements a general-purpose AVL (balanced) tree class: the TAVLTree (61) class and it's associated data node class TAVLTreeNode (70).

2.3 TAVLTree

2.3.1 Description

`TAVLTree` maintains a balanced AVL tree. The tree consists of TAVLTreeNode (70) 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 (69) property.

2.3.2 Method overview

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

2.3.3 Property overview

Page	Property	Access	Description
70	Count	r	Number of nodes in the tree.
69	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 (69) 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 (64).

See also: TAVLTree.OnCompare (69), TAVLTree.FindKey (63)

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: TAVLTree.OnCompare (69), TAVLTree.Find (62)

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 (63), TAVLTree.MoveDataRightMost (67)

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 (63), TAVLTree.MoveDataLeftMost (67)

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: TAVLTree.FindHighest (64)

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: TAVLTree.FindLowest (63)

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: TAVLTree.FindHighest (64), TAVLTree.FindLowest (63), TAVLTree.Find (62), TAVLTree.FindKey (63)

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

The default OnCompare (69) compares the data pointers, so the default `Find` will return the same node as `FindPointer`

See also: TAVLTree.Find (62), TAVLTree.FindKey (63)

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 (65), TAVLTree.FindLeftMostKey (65), TAVLTree.FindRightMostKey (65)

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 succeding 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 (64), TAVLTree.FindLeftMostKey (65), TAVLTree.FindRightMostKey (65)

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 (64), TAVLTree.FindRightMost (65), TAVLTree.FindRightMostKey (65)

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 succeding node for `Key` and tries to move as far right in the tree as possible.

See also: TAVLTree.FindLeftMost (64), TAVLTree.FindRightMost (65), TAVLTree.FindLeftMostKey (65)

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

See also: TAVLTree.FindLeftMost (64), TAVLTree.FindLeftMostKey (65), TAVLTree.FindRightMostSameKey (66)

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

See also: TAVLTree.FindRightMost (65), TAVLTree.FindRightMostKey (65), TAVLTree.FindLeftMostSameKey (65)

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 (66), TAVLTree.Remove (66)

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 (70) 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 (66), TAVLTree.RemovePointer (67), TAVLTree.Clear (67)

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

See also: TAVLTree.Delete (66), TAVLTree.RemovePointer (67), TAVLTree.Clear (67), TAVLTree.Find (62)

2.3.21 TAVLTree.RemovePointer

Synopsis: Remove a pointer item from the list.

Declaration: `procedure RemovePointer(Data: Pointer)`

Visibility: public

Description: `Remove` uses FindPointer (64) 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 (66), TAVLTree.Delete (66), TAVLTree.Clear (67)

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 (67)

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 (67)

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 (68) instead.

See also: TAVLTree.FreeAndClear (68), TAVLTree.Delete (66)

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 (67), TAVLTree.Delete (66), TAVLTree.FreeAndDelete (68)

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 (67), TAVLTree.Delete (66), TAVLTree.FreeAndClear (68)

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 contais 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 (68)

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 returnes in `StreamSize`. This method is only useful for debugging purposes.

See also: TAVLTree.ConsistencyCheck (68)

2.3.29 TAVLTree.ReportAsString

Synopsis: Return the tree report as a string

Declaration: `function ReportAsString : String`

Visibility: public

Description: `ReportAsString` calls WriteReportToStream (68) and retuns the stream data as a string.

See also: TAVLTree.WriteReportToStream (68)

2.3.30 TAVLTree.SetNodeManager

Declaration: `procedure SetNodeManager(newmgr: TBaseAVLTreeNodeManager)`

Visibility: public

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 (61). An alternate OnCompare (61) can be provided: the default `OnCompare` method compares the 2 data pointers of a node.

See also: OnCompare (61)

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 (69), TAVLTree.Clean (61)

2.3.33 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 (69)

2.3.34 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.

2.4.2 Method overview

Page	Property	Description
70	Clear	Clears the node's data
70	TreeDepth	Level of the node in the tree below

2.4.3 TAVLTreeNode.Clear

Synopsis: Clears the node's data

Declaration: `procedure Clear`

Visibility: public

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

2.4.4 TAVLTreeNode.TreeDepth

Synopsis: Level of the node in the tree below

Declaration: `function TreeDepth : Integer`

Visibility: public

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

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

2.5 TAVLTreeNodeMemManager

2.5.1 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.

2.5.2 Method overview

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

2.5.3 Property overview

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

2.5.4 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 (71)

2.5.5 TAVLTreeNodeMemManager.NewNode

Synopsis: Create a new `TAVLTreeNode` instance

Declaration: `function NewNode : TAVLTreeNode; Override`

Visibility: public

Description: `NewNode` returns a new TAVLTreeNode (70) instance. If there is a node in the free list, itare returned. If no more free nodes are present, a new node is created.

See also: TAVLTreeNodeMemManager.DisposeNode (71)

2.5.6 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 (72), TAVLTreeNodeMemManager.MaximumFreeNodeRatio (72)

2.5.7 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 (72)

2.5.8 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 (72)

2.5.9 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 (72)

2.5.10 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 (73), TAVLTreeNodeMemManager.MinimumFreeNode (72)

2.5.11 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 (72), TAVLTreeNodeMemManager.MaximumFreeNodeRatio (72)

2.6 TBaseAVLTreeNodeManager

2.6.1 Method overview

Page	Property	Description
73	DisposeNode	
73	NewNode	

2.6.2 TBaseAVLTreeNodeManager.DisposeNode

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

Visibility: public

2.6.3 TBaseAVLTreeNodeManager.NewNode

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

Visibility: public

Chapter 3

Reference for unit 'base64'

3.1 Used units

Table 3.1: Used units by unit 'base64'

Name	Page
Classes	**??**
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 (77) for encoding, and TBase64DecodingStream (75) for decoding.

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

3.3 Constants, types and variables

3.3.1 Types

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

Table 3.2: Enumeration values for type TBase64DecodingMode

Value	Explanation
bdmMIME	MIME encoding
bdmStrict	Strict encoding

`TBase64DecodingMode` determines the decoding algorithm used by TBase64DecodingStream (75). 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 (75) 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 EBase64DecodingException

3.4.1 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.

3.5 TBase64DecodingStream

3.5.1 Description

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

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

3.5.2 Method overview

Page	Property	Description
75	Create	Create a new instance of the `TBase64DecodingStream` class
76	Read	Read and decrypt data from the source stream
76	Reset	Reset the stream
76	Seek	Set stream position.

3.5.3 Property overview

Page	Property	Access	Description
77	EOF	r	
77	Mode	rw	Decoding mode

3.5.4 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 (77), TBase64DecodingMode (74)

3.5.5 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 (77), TBase64DecodingStream.Read (76)

3.5.6 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 (75), TBase64DecodingStream.Seek (76), #rtl.classes.TStream.Read (**??**)

3.5.7 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 (76), TBase64DecodingStream.Write (75), TBase64EncodingStream.Seek (78), #rtl.classes.TStream.Seek (**??**)

3.5.8 TBase64DecodingStream.EOF

Synopsis:

Declaration: `Property EOF : Boolean`

Visibility: public

Access: Read

Description:

3.5.9 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 (75)

3.6 TBase64EncodingStream

3.6.1 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.

3.6.2 Method overview

Page	Property	Description
77	Create	Create a new instance of the `TBase64EncodingStream` class.
78	Destroy	Remove a `TBase64EncodingStream` instannce from memory
78	Seek	Position the stream
78	Write	Write data to the stream.

3.6.3 TBase64EncodingStream.Create

Synopsis: Create a new instance of the `TBase64EncodingStream` class.

Declaration: `constructor Create(ASource: TStream)`

Visibility: public

Description: `Create` instantiates a new `TBase64EncodingStream` class. The `ASource` stream is stored and used to write the encoded data to.

See also: TBase64EncodingStream.Destroy (78), TBase64DecodingStream.Create (75)

3.6.4 TBase64EncodingStream.Destroy

Synopsis: Remove a `TBase64EncodingStream` instannce 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 (77)

3.6.5 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 (78), TBase64EncodingStream.Read (77), TBase64DecodingStream.Write (75), #rtl.classes.TStream.Write (**??**)

3.6.6 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 (77), TBase64EncodingStream.Write (78), #rtl.classes.TStream.Seek (**??**)

Chapter 4

Reference for unit 'BlowFish'

4.1 Used units

Table 4.1: Used units by unit 'BlowFish'

Name	Page
Classes	??
sysutils	??

4.2 Overview

The BlowFish implements a class TBlowFish (80) to handle blowfish encryption/descryption of memory buffers, and 2 TStream (**??**) descendents TBlowFishDeCryptStream (81) which descrypts any data that is read from it on the fly, as well as TBlowFishEnCryptStream (82) 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 (80) array.

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

TBFBlock is the basic data structure used by the encrypting/decrypting routines in TBlowFish (80), TBlowFishDeCryptStream (81) and TBlowFishEnCryptStream (82). 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 (80), TBlowFishDeCryptStream (81) and TBlowFishEnCryptStream (82) 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 (83), TBlowFishEncryptStream (82) and TBlowFishDecryptStream (81) classes to report errors.

4.5 TBlowFish

4.5.1 Description

TBlowFish is a simple class that can be used to encrypt/decrypt a single TBFBlock (80) data block with the Encrypt (80) and Decrypt (81) calls. It is used internally by the TBlowFishEnCryptStream (82) and TBlowFishDeCryptStream (81) classes to encrypt or decrypt the actual data.

4.5.2 Method overview

Page	Property	Description
80	Create	Create a new instance of the TBlowFish class
81	Decrypt	Decrypt a block
80	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 (80) and Decrypt (81).

See also: TBlowFish.Encrypt (80), TBlowFish.Decrypt (81)

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

See also: TBlowFishKey (80), TBlowFish.Decrypt (81), TBlowFish.Create (80)

4.5.5 TBlowFish.Decrypt

Synopsis: Decrypt a block

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

Visibility: public

Description: `ncrypt` decrypts the data in `Block` (always 8 bytes) using the key (80) specified when the `TBlowFish` instance was created. The data must have been encrypted with the same key and the Encrypt (80) call.

See also: TBlowFishKey (80), TBlowFish.Encrypt (80), TBlowFish.Create (80)

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 (82) stream must be used.

4.6.2 Method overview

Page	Property	Description
81	Read	Read data from the stream
82	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: TBlowFishStream.Create (84), TBlowFishEncryptStream (82)

4.6.4 TBlowFishDeCryptStream.Seek

Synopsis: Set the stream position.

Declaration: `function Seek(Offset: LongInt;Origin: Word) : LongInt;  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 (80) exception will be raised.

See also: TBlowFishDeCryptStream.Read (81), EBlowFishError (80)

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 (81) stream must be used.

4.7.2 Method overview

Page	Property	Description
82	Destroy	Free the `TBlowFishEncryptStream`
83	Flush	Flush the encryption buffer
83	Seek	Set the position in the stream
82	Write	Write data to the stream

4.7.3 TBlowFishEncryptStream.Destroy

Synopsis: Free the `TBlowFishEncryptStream`

Declaration: `destructor Destroy;  Override`

Visibility: public

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

See also: TBlowFishEncryptStream.Flush (83), TBlowFishStream.Create (84)

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

See also: TBlowFishEncryptStream.Read (82)

4.7.5 TBlowFishEncryptStream.Seek

Synopsis: Set the position in the stream

Declaration: `function Seek(Offset: LongInt;Origin: Word) : LongInt;  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 (80) exception.

See also: TBlowFishEncryptStream.Write (82)

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: TBlowFishEncryptStream.Write (82), TBFBlock (80)

4.8 TBlowFishStream

4.8.1 Description

`TBlowFishStream` is an abstract class which is used as a parent class for TBlowFishEncrypt-Stream (82) and TBlowFishDecryptStream (81). It simply provides a constructor and storage for a TBlowFish (80) 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`.

4.8.2 Method overview

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

4.8.3 Property overview

Page	Property	Access	Description
84	BlowFish	r	Blowfish instance used when encrypting/descrypting

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 (80) 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 (82), TBlowFishDecryptStream (81), TBlowFish (80)

4.8.5 TBlowFishStream.Destroy

Synopsis: Destroy the `TBlowFishStream` instance.

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

Visibility: public

Description: `Destroy` cleans up the internal TBlowFish (80) instance.

Errors:

See also: TBlowFishStream.Create (84), TBlowFish (80)

4.8.6 TBlowFishStream.BlowFish

Synopsis: Blowfish instance used when encrypting/descrypting

Declaration:
```
Property BlowFish : TBlowFish
```

Visibility: public

Access: Read

Description: `BlowFish` is the TBlowFish (80) instance which is created when the `TBlowFishStream` class is initialized. Normally it should not be used directly, it's intended for access by the descencent classes TBlowFishEncryptStream (82) and TBlowFishDecryptStream (81).

See also: TBlowFishEncryptStream (82), TBlowFishDecryptStream (81), TBlowFish (80)

Chapter 5

Reference for unit 'bufstream'

5.1 Used units

Table 5.1: Used units by unit 'bufstream'

Name	Page
Classes	??
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 (89) which is for reading only, and TWriteBufStream (89) 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 (89) and TWriteBufStream (89) 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.

5.4.2 Method overview

Page	Property	Description
87	Create	Create a new TBufStream instance.
87	Destroy	Destroys the TBufStream instance

5.4.3 Property overview

Page	Property	Access	Description
87	Buffer	r	The current buffer
88	BufferPos	r	Current buffer position.
88	BufferSize	r	Amount of data in the buffer
88	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 (86) is used as the capacity.

An instance of TBufStream should never be instantiated directly. Instead, an instance of TReadBufStream (89) or TWriteBufStream (89) should be created.

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

See also: TBufStream.Destroy (87), TReadBufStream (89), TWriteBufStream (89)

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 (87), TReadBufStream (89), TWriteBufStream (89)

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 (87), TBufStream.Capacity (88), TBufStream.BufferSize (88)

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 (87), TBufStream.Buffer (87), TBufStream.BufferSize (88), TBufStream.BufferPos (88)

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 (87), TBufStream.Buffer (87), TBufStream.BufferSize (88), TBufStream.Capacity (88)

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 (88).

See also: TBufStream.Create (87), TBufStream.Buffer (87), TBufStream.BufferPos (88), TBufStream.Capacity (88)

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.

5.5.2 Method overview

Page	Property	Description
89	Read	Reads data from the stream
89	Seek	Set location in the buffer

5.5.3 TReadBufStream.Seek

Synopsis: Set location in the buffer

Declaration: `function Seek(Offset: LongInt;Origin: Word) : LongInt;  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 (90), TReadBufStream.Read (89), TReadBufStream.Write (89)

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 untill `ACount` bytes are read, or no more data is available.

See also: TReadBufStream.Seek (89), TReadBufStream.Read (89)

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.

5.6.2 Method overview

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

5.6.3 TWriteBufStream.Destroy

Synopsis: Remove the `TWriteBufStream` instance from memory

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

Visibility: public

Description: `Destroy` flushes the buffer and then calls the inherited Destroy (87).

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

See also: TBufStream.Create (87), TBufStream.Destroy (87)

5.6.4 TWriteBufStream.Seek

Synopsis: Set stream position.

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

Errors:

See also: TWriteBufStream.Write (90), TWriteBufStream.Read (89), TReadBufStream.Seek (89)

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 (90), TWriteBufStream.Read (89), TReadBufStream.Write (89)

Chapter 6

Reference for unit 'CacheCls'

6.1 Used units

Table 6.1: Used units by unit 'CacheCls'

Name	Page
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 calles `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 (92) record.

```
PCacheSlotArray = ^TCacheSlotArray
```

Pointer to TCacheSlotArray (92) array

```
TCacheSlot = record
```

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

TCacheSlot is internally used by the TCache (92) 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 (91) 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 (93) 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 (95) property offers indexed access to the array of items. When accessing the array through this property, the MRUSlot (95) property is updated.

6.5.2 Method overview

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

6.5.3 Property overview

Page	Property	Access	Description
95	Data	rw	Indexed access to data items
96	LRUSlot	r	Last used item
95	MRUSlot	rw	Most recent item slot.
97	OnFreeSlot	rw	Event called when a slot is freed
96	OnIsDataEqual	rw	Event to compare 2 items.
96	SlotCount	rw	Number of slots in the list
96	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 (96)

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 (93)

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 (94) 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 (94), TCache.FindSlot (94), TCache.IndexOf (94), TCache.Data (95), TCache.MRUSlot (95)

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 (93) 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 (93), TCache.FindSlot (94), TCache.IndexOf (94), TCache.Data (95), TCache.MRUSlot (95)

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 (96) event must be set.

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

See also: TCache.IndexOf (94), TCache.Add (93), TCache.OnIsDataEqual (96)

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 (96) event must be set.

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

See also: TCache.FindSlot (94), TCache.Add (93), TCache.OnIsDataEqual (96)

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 (94)

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

The index is zero-based and can maximally be SlotCount-1 (96). Providing an invalid index will result in an exception.

See also: TCache.MRUSlot (95)

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 (95) property, or when an item is added to the list with Add (93) or AddNew (94)

See also: TCache.Add (93), TCache.AddNew (94), TCache.Data (95), TCache.LRUSlot (96)

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 (93), TCache.AddNew (94), TCache.Data (95), TCache.MRUSlot (95)

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 (97) is called.

See also: TCache.Create (93), TCache.Data (95), TCache.Slots (96)

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 (96). Providing an invalid index will result in an exception.

See also: TCache.Data (95), TCache.SlotCount (96)

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 (94) and IndexOf (94) to compare items when looking for a particular item. These functions are called by the Add (93) 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 (94), TCache.IndexOf (94), TCache.Add (93)

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 (93), TCache.AddNew (94), TCache.SlotCount (96)

Chapter 7

Reference for unit 'contnrs'

7.1 Used units

Table 7.1: Used units by unit 'contnrs'

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

7.2 Overview

The contnrs unit implements various general-purpose classes:

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

Component lists lists that manage components instead of pointers, and which automatically dispose 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 div16
```

`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 div4
```

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 (99)" type.

```
PHashItem = ^THashItem
```

PHashItem is a pointer type, pointing to the THashItem (101) record.

```
PHashItemList = ^THashItemList
```

PHashItemList is a pointer to the THashItemList (101). It's used in the TFPHashList (118) as a pointer to the memory area containing the hash item records.

```
PHashTable = ^THashTable
```

PHashTable is a pointer to the THashTable (101). It's used in the TFPHashList (118) 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 (109) class. It is a container for TBucketItem (100) records. It should never be used directly.

```
TBucketArray = Array of TBucket
```

Array of TBucket (99) records.

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

`TBucketItem` is a record used for internal use in TCustomBucketList (109). 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 (102).

```
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 (100) 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 (100) prototype.

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

`TDataIteratorMethod` is a callback prototype for the TDataHashTable.Iterate (98) 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 (99) type. It's used in the TFPHashList (118) 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 (99) class.

```
THTCustomNodeClass = Class of THTCustomNode
```

THTCustomNodeClass is used by THTCustomHashTable (98) 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 TFPHashTable (98) 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 (142) 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 (142) 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 (111) 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 (111) hash class. It's Robert Sedgwick's "Algorithms in C" hash function.

Errors: None.

See also: TFPCustomHashTable (111)

7.5 EDuplicate

7.5.1 Description

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

7.6 EKeyNotFound

7.6.1 Description

Exception raised when a key is not found.

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.

7.7.2 Method overview

Page	Property	Description
103	Create	Create a new `TBucketList` instance.

7.7.3 TBucketList.Create

Synopsis: Create a new `TBucketList` instance.

Declaration: `constructor Create(ABuckets: TBucketListSizes)`

Visibility: public

Description: `Create` instantiates a new bucketlist instance with a number of buckets determined by `ABuckets`. After creation, the number of buckets can no longer be changed.

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

See also: TBucketListSizes (100)

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.

7.8.2 Method overview

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

7.8.3 Property overview

Page	Property	Access	Description
105	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 (**??**) bevahiour, and introduces no new functionality.

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

See also: TClassList.Extract (104), #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 (104), #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 (104), #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 (105), TClassList.Pack (103)

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 (105), TClassList.Pack (103)

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 (103), TClassList.Remove (104)

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

7.9.2 Method overview

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

7.9.3 Property overview

Page	Property	Access	Description
108	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 (149), #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: TObectList.Add (98)

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 (149) 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 (107), TObjectList.Extract (150)

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

Errors: None.

See also: TComponentList.Extract (107), TObjectList.Remove (151)

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 (151)

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 (108), TObjectList.First (152)

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 (108), TObjectList.Last (152)

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 (152), TComponentList.Add (106), TComponentList.Remove (107)

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 (153)

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, escpecially 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 (102). `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 (109).

7.10.2 Method overview

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

7.10.3 Property overview

Page	Property	Access	Description
111	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: `Detsroy` 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 (110)

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 (110), TCustomBucketList.Clear (109)

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 (110), TCustomBucketList.Clear (109)

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 returne5Ad.

If the data pointer associated with `AItem` is also needed, then it is better to use Find (110).

See also: TCustomBucketList.Find (110)

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 (110)

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 (111)

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: TCustomBucketList.Find (110)

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 (110), TCustomBucketList.Exists (110)

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 (102).

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.

7.11.2 Method overview

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

7.11.3 Property overview

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

7.11.4 TFPCustomHashTable.Create

Synopsis: Instantiate a new `TFPCustomHashTable` instance using the default hash mechanism

Declaration: `constructor Create`

Visibility: public

Description: `Create` creates a new instance of `TFPCustomHashTable` with hash size 196613 and hash algorithm RSHash (102)

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

See also: TFPCustomHashTable.CreateWith (112)

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: TFPCustomHashTable.Create (112)

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: TFPCustomHashTable.Destroy (113), TFPCustomHashTable.Create (112), TFPCustomHashTable.CreateWith (112), THTCustomNode.Data (145)

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: TFPCustomHashTable.HashTableSize (115)

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: TFPCustomHashTable.Destroy (113)

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 (114), TFPCustomHashTable.Add (111)

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 (145) 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 (114) method should be used instead.

Errors: None.

See also: TFPCustomHashTable.Add (111), TFPCustomHashTable.Delete (114)

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.

Errors:

See also: TFPCustomHashTable.Count (115), TFPCustomHashTable.HashTableSize (115), TFPCustomHashTable.AVGChainLen (116), TFPCustomHashTable.MaxChainLength (116)

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 (111). 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: TFPCustomHashTable.SetHashFunction (111), TFPCustomHashTable.HashTableSize (115)

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 (114), TFPCustomHashTable.HashTableSize (115), TFPCustomHashTable.AVGChainLen (116), TFPCustomHashTable.MaxChainLength (116)

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 (114), TFPCustomHashTable.Count (115), TFPCustomHashTable.AVGChainLen (116), TFPCustomHashTable.MaxChainLength (116), TFPCustomHashTable.VoidSlots (116), TFPCustomHashTable.Density (117)

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 (137) used for the hash table. Each element in this table is again a TFPObjectList (137) 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 (114), TFPCustomHashTable.Count (115), TFPCustomHashTable.AVGChainLen (116), TFPCustomHashTable.MaxChainLength (116), TFPCustomHashTable.LoadFactor (116), TFPCustomHashTable.Density (117), TFPCustomHashTable.NumberOfCollisions (117)

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 (114), TFPCustomHashTable.Count (115), TFPCustomHashTable.AVGChainLen (116), TFPCustomHashTable.MaxChainLength (116), TFPCustomHashTable.VoidSlots (116), TFPCustomHashTable.Density (117), TFPCustomHashTable.NumberOfCollisions (117)

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 (114), TFPCustomHashTable.Count (115), TFPCustomHashTable.LoadFactor (116), TFPCustomHashTable.MaxChainLength (116), TFPCustomHashTable.VoidSlots (116), TFPCustomHashTable.Density (117), TFPCustomHashTable.NumberOfCollisions (117)

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 (114), TFPCustomHashTable.Count (115), TFPCustomHashTable.LoadFactor (116), TFPCustomHashTable.AvgChainLength (111), TFPCustomHashTable.VoidSlots (116), TFPCustomHashTable.Density (117), TFPCustomHashTable.NumberOfCollisions (117)

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 (114), TFPCustomHashTable.Count (115), TFPCustomHashTable.LoadFactor (116), TFPCustomHashTable.AvgChainLength (111), TFPCustomHashTable.VoidSlots (116), TFPCustomHashTable.Density (117)

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 (114), TFPCustomHashTable.Count (115), TFPCustomHashTable.LoadFactor (116), TFPCustomHashTable.AvgChainLength (111), TFPCustomHashTable.VoidSlots (116), TFPCustomHashTable.Density (117)

7.12 TFPDataHashTable

7.12.1 Description

`TFPDataHashTable` is a TFPCustomHashTable (111) 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 (135), or for string data, TFPStringHashTable (144) is more suitable. The data pointers are exposed with their keys through the Items (118) property.

7.12.2 Method overview

Page	Property	Description
118	Add	Add a data pointer to the list.

7.12.3 Property overview

Page	Property	Access	Description
118	Items	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 (118)

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 (144)

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 (111) class can be used. If a hash class for objects is needed instead of pointers, the TFPHashObjectList (128) class can be used.

7.13.2 Method overview

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

7.13.3 Property overview

Page	Property	Access	Description
124	Capacity	rw	Capacity of the list.
124	Count	rw	Current number of elements in the list.
125	Items	rw	Indexed array with pointers
125	List	r	Low-level hash list
125	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 (119)

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 (119), TFPHashList.Clear (120)

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 (122), TFPHashList.Remove (123), TFPHashList.Delete (121)

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 (122), TFPHashList.Remove (123), TFPHashList.Delete (121), TFPHashList.Add (120)

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 (120), TFPHashList.Find (122), TFPHashList.FindIndexOf (122), TFPHashList.FindWithHash (123)

7.13.9 TFPHashList.HashOfIndex

Synopsis: Return the hash valye 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 (118), TFPHashList.Find (122), TFPHashList.FindIndexOf (122), TFPHashList.FindWithHash (123)

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

Errors:

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 (122)TFPHashList.Remove (123)TFPHashList.Add (120)

7.13.12 TFPHashList.Error

Synopsis: Raise an error

Declaration: `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 (120)

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 (121), TFPHashList.Remove (123), TFPHashList.Clear (120)

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 (120), TFPHashList.NameOfIndex (120), TFPHashList.Find (122), TFPHashList.FindIndexOf (122), TFPHashList.FindWithHash (123)

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 (120), TFPHashList.NameOfIndex (120), TFPHashList.IndexOf (122), TFPHashList.FindIndexOf (122), TFPHashList.FindWithHash (123)

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 (120), TFPHashList.NameOfIndex (120), TFPHashList.IndexOf (122), TFPHashList.Find (122), TFPHashList.FindWithHash (123)

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 (120), TFPHashList.NameOfIndex (120), TFPHashList.IndexOf (122), TFPHashList.Find (122), TFPHashList.FindIndexOf (122)

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 occurence 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 (121), TFPHashList.Clear (120), TFPHashList.Extract (122)

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 (120)

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: TFPHashList.Count (124), TFPHashList.Items (125)

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: TFPHashList.Capacity (124), TFPHashList.Items (125)

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 (124).

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

See also: TFPHashList.Capacity (124), TFPHashList.Count (124)

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 (101). It should not be used directly.

See also: TFPHashList.Strs (125), THashItemList (101)

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: TFPHashList.List (125)

7.14 TFPHashObject

7.14.1 Description

`TFPHashObject` is a `TObject` descendent which is aware of the TFPHashObjectList (128) 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.

7.14.2 Method overview

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

7.14.3 Property overview

Page	Property	Access	Description
127	Hash	r	Hash value
127	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 (128) hash list. It also has no name when created in this way.

See also: TFPHashObject.Name (127), TFPHashObject.ChangeOwner (126), TFPHashObject.ChangeOwnerAndName (127)

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 (128).

See also: TFPHashObject.CreateNotOwned (126), TFPHashObject.ChangeOwner (126), TFPHashObject.Name (127)

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: TFPHashObject.ChangeOwnerAndName (127), TFPHashObject.Name (127)

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: TFPHashObject.ChangeOwner (126), TFPHashObject.Name (127)

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: TFPHashObject.ChangeOwner (126), TFPHashObject.ChangeOwnerAndName (127), TFPHashObject.Name (127)

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: TFPHashObject.Rename (127), TFPHashObject.ChangeOwnerAndName (127)

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: TFPHashObject.Name (127)

7.15 TFPHashObjectList

7.15.1 Method overview

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

7.15.2 Property overview

Page	Property	Access	Description
133	Capacity	rw	Capacity of the list.
134	Count	rw	Current number of elements in the list.
134	Items	rw	Indexed array with object instances
134	List	r	Low-level hash list
134	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 (128), TFPHashObjectList.OwnsObjects (134)

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 (128), TFPHashObjectList.Clear (129)

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

Errors: None.

See also: TFPHashObjectList.Extract (131), TFPHashObjectList.Remove (131), TFPHashObjectList.Delete (130), TFPHashObjectList.Add (129)

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 (131), TFPHashObjectList.Remove (131), TFPHashObjectList.Delete (130)

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 (130), TFPHashObjectList.Find (131), TFPHashObjectList.FindIndexOf (132), TFPHashObjectList.FindWithHash (132)

7.15.8 TFPHashObjectList.HashOfIndex

Synopsis: Return the hash valye 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 (128), TFPHashObjectList.Find (131), TFPHashObjectList.FindIndexOf (132), TFPHashObjectList.FindWithHash (132)

7.15.9 TFPHashObjectList.GetNextCollision

Synopsis: Get next collision number

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

Visibility: public

Description: Get next collision number

Errors:

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

See also: TFPHashObjectList.Extract (131), TFPHashObjectList.Remove (131), TFPHashObjectList.Add (129), TFPHashObjectList.OwnsObjects (134)

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 (129)

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 (134).

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

See also: TFPHashObjectList.Delete (130), TFPHashObjectList.Remove (131), TFPHashObjectList.Clear (129)

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 occurence 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 (134) is `True`, then the object itself is also freed from memory.

See also: TFPHashObjectList.Delete (130), TFPHashObjectList.Clear (129), TFPHashObjectList.Extract (131)

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 (130), TFPHashObjectList.NameOfIndex (129), TFPHashObjectList.Find (131), TFPHashObjectList.FindIndexOf (132), TFPHashObjectList.FindWithHash (132)

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 (130), TFPHashObjectList.NameOfIndex (129), TFPHashObjectList.IndexOf (131), TFPHashObjectList.FindIndexOf (132), TFPHashObjectList.FindWithHash (132)

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 (130), TFPHashObjectList.NameOfIndex (129), TFPHashObjectList.IndexOf (131), TFPHashObjectList.Find (131), TFPHashObjectList.FindWithHash (132)

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 (130), TFPHashObjectList.NameOfIndex (129), TFPHashObjectList.IndexOf (131), TFPHashObjectList.Find (131), TFPHashObjectList.FindIndexOf (132)

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 (129)

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: TFPHashObjectList.Count (134), TFPHashObjectList.Items (134)

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: TFPHashObjectList.Capacity (133), TFPHashObjectList.Items (134)

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 (128)

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 (134).

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

See also: TFPHashObjectList.Capacity (133), TFPHashObjectList.Count (134)

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 (118). It should not be used directly.

See also: TFPHashList (118)

7.16 TFPObjectHashTable

7.16.1 Description

`TFPStringHashTable` is a TFPCustomHashTable (111) 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 (144), or for arbitrary pointer data, TFPDataHashTable (117) is more suitable. The objects are exposed with their keys through the Items (136) property.

7.16.2 Method overview

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

7.16.3 Property overview

Page	Property	Access	Description
136	Items	rw	Key-based access to the objects
136	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 (136) 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 (136), TFPObjectHashTable.CreateWith (135), TFPObjectHashTable.Items (136)

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 (136) property to `AOwnsObjects`, and then calls the inherited `CreateWith`. If `AOwnsObjects` is set to `True`, then the hash table owns the objects: whenever an object is removed from the list, it is automatically freed.

This constructor should be used when a table size and hash algorithm should be specified that differ from the default table size and hash algorithm.

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

See also: TFPObjectHashTable.OwnsObjects (136), TFPObjectHashTable.Create (135), TFPObjectHashTable.Items (136)

7.16.6 TFPObjectHashTable.Add

Synopsis: Add a new object to the hash table

Declaration: `procedure Add(const aKey: String;AItem: TObject);  Virtual`

Visibility: public

Description: `Add` adds the object `AItem` to the hash table, and associates it with key `aKey`.

Errors: If the key `aKey` is already in the hash table, an exception will be raised.

See also: TFPObjectHashTable.Items (136)

7.16.7 TFPObjectHashTable.Items

Synopsis: Key-based access to the objects

Declaration: `Property Items[index: String]: TObject; default`

Visibility: public

Access: Read,Write

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

See also: TFPObjectHashTable.Add (136)

7.16.8 TFPObjectHashTable.OwnsObjects

Synopsis: Does the hash table own the objects ?

Declaration: `Property OwnsObjects : Boolean`

Visibility: public

Access: Read,Write

Description: `OwnsObjects` determines what happens with objects which are removed from the hash table: if `True`, then removing an object from the hash list will free the object. If `False`, the object is not freed. Note that way in which the object is removed is not relevant: be it `Delete`, `Remove` or `Clear`.

See also: TFPObjectHashTable.Create (135), TFPObjectHashTable.Items (136)

7.17 TFPObjectList

7.17.1 Description

`TFPObjectList` is a TFPList (**??**) based list which has as the default array property TObjects (**??**) instead of pointers. By default it also manages the objects: when an object is deleted or removed from the list, it is automatically freed. This behaviour can be disabled when the list is created.

In difference with TObjectList (149), `TFPObjectList` offers no notification mechanism of list operations, allowing it to be faster than `TObjectList`. For the same reason, it is also not a descendent of `TFPList` (although it uses one internally).

7.17.2 Method overview

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

7.17.3 Property overview

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

7.17.4 TFPObjectList.Create

Synopsis: Create a new object list

Declaration:
```
constructor Create
constructor Create(FreeObjects: Boolean)
```

Visibility: public

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

Errors: None.

See also: TFPObjectList.Destroy (138), TFPObjectList.OwnsObjects (143), TObjectList (149)

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 (143) is `True`.

See also: TFPObjectList.OwnsObjects (143), TObjectList.Create (150)

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 (143) is `True`.

See also: TObjectList.Destroy (149)

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 (143) 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 (143), TFPObjectList.Delete (138)

7.17.8 TFPObjectList.Delete

Synopsis: Delete an element from the list.

Declaration: `procedure Delete(Index: Integer)`

Visibility: public

Description: `Delete` removes the object at index `Index` from the list. When OwnsObjects (143) is `True`, the object is also freed.

Errors: An access violation may occur when OwnsObjects (143) is `True` and either the object was freed externally, or when the same object is in the same list twice.

See also: TTFPObjectList.Remove (98), TFPObjectList.Extract (139), TFPObjectList.OwnsObjects (143), TTFPObjectList.Add (98), TTFPObjectList.Clear (98)

7.17.9 TFPObjectList.Exchange

Synopsis: Exchange the location of two objects

Declaration: `procedure Exchange(Index1: Integer;Index2: Integer)`

Visibility: public

Description: `Exchange` exchanges the objects at indexes `Index1` and `Index2` in a direct operation (i.e. no delete/add is performed).

Errors: If either `Index1` or `Index2` is invalid, an exception will be raised.

See also: TTFPObjectList.Add (98), TTFPObjectList.Delete (98)

7.17.10 TFPObjectList.Expand

Synopsis: Expand the capacity of the list.

Declaration: `function Expand : TFPObjectList`

Visibility: public

Description: `Expand` increases the capacity of the list. It calls #rtl.classes.tfplist.expand (**??**) and then returns a reference to itself.

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

See also: TFPObjectList.Pack (142), TFPObjectList.Clear (138), #rtl.classes.tfplist.expand (**??**)

7.17.11 TFPObjectList.Extract

Synopsis: Extract an object from the list

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

Visibility: public

Description: `Extract` removes `Item` from the list, if it is present in the list. It returns `Item` if it was found, `Nil` if item was not present in the list.

Note that the object is not freed, and that only the first found object is removed from the list.

Errors: None.

See also: TFPObjectList.Pack (142), TFPObjectList.Clear (138), TFPObjectList.Remove (140), TFPObjectList.Delete (138)

7.17.12 TFPObjectList.Remove

Synopsis: Remove an item from the list.

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

Visibility: public

Description: `Remove` removes `Item` from the list, if it is present in the list. It frees `Item` if OwnsObjects (143) is `True`, and returns the index of the object that was found in the list, or -1 if the object was not found.

Note that only the first found object is removed from the list.

Errors: None.

See also: TFPObjectList.Pack (142), TFPObjectList.Clear (138), TFPObjectList.Delete (138), TFPObjectList.Extract (139)

7.17.13 TFPObjectList.IndexOf

Synopsis: Search for an object in the list

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

Visibility: public

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

Errors: None.

See also: TFPObjectList.Items (144), TFPObjectList.Remove (140), TFPObjectList.Extract (139)

7.17.14 TFPObjectList.FindInstanceOf

Synopsis: Search for an instance of a certain class

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

Visibility: public

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

If no instance of the requested class is found, `Nil` is returned.

Errors: None.

See also: TFPObjectList.IndexOf (140)

7.17.15 TFPObjectList.Insert

Synopsis: Insert a new object in the list

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

Visibility: public

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

Errors: None.

See also: TFPObjectList.Add (138), TFPObjectList.Delete (138)

7.17.16 TFPObjectList.First

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

Declaration: `function First : TObject`

Visibility: public

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

Errors: None.

See also: TFPObjectList.Last (141), TFPObjectList.Pack (142)

7.17.17 TFPObjectList.Last

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

Declaration: `function Last : TObject`

Visibility: public

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

Errors: None.

See also: TFPObjectList.First (141), TFPObjectList.Pack (142)

7.17.18 TFPObjectList.Move

Synopsis: Move an object to another location in the list.

Declaration: `procedure Move(CurIndex: Integer;NewIndex: Integer)`

Visibility: public

Description: `Move` moves the object at current location `CurIndex` to location `NewIndex`. Note that the `NewIndex` is determined *after* the object was removed from location `CurIndex`, and can hence be shifted with 1 position if `CurIndex` is less than `NewIndex`.

Contrary to exchange (139), 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 (139), TFPObjectList.Delete (138), TFPObjectList.Insert (140)

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 (141), TFPObjectList.Last (141)

7.17.21 TFPObjectList.Sort

Synopsis: Sort the list of objects

Declaration: `procedure Sort(Compare: TListSortCompare)`

Visibility: public

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

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

equal to 0 If the pointers have the same value.

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

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

Errors: None.

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

7.17.22 TFPObjectList.ForEachCall

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

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

Visibility: public

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

Errors: None.

See also: TObjectListStaticCallback (102), TObjectListCallback (101)

7.17.23 TFPObjectList.Capacity

Synopsis: Capacity of the list

Declaration:
```
Property Capacity : Integer
```

Visibility: public

Access: Read,Write

Description: `Capacity` is the number of elements that the list can contain before it needs to expand itself, i.e., reserve more memory for pointers. It is always equal or larger than Count (143).

See also: TFPObjectList.Count (143)

7.17.24 TFPObjectList.Count

Synopsis: Number of elements in the list.

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

Visibility: public

Access: Read,Write

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

See also: TFPObjectList.Capacity (143)

7.17.25 TFPObjectList.OwnsObjects

Synopsis: Should the list free elements when they are removed.

Declaration:
```
Property OwnsObjects : Boolean
```

Visibility: public

Access: Read,Write

Description: `OwnsObjects` determines whether the objects in the list should be freed when they are removed (not extracted) from the list, or when the list is cleared. If the property is `True` then they are freed. If the property is `False` the elements are not freed.

The value is usually set in the constructor, and is seldom changed during the lifetime of the list. It defaults to `True`.

See also: TFPObjectList.Create (137), TFPObjectList.Delete (138), TFPObjectList.Remove (140), TFPObjectList.Clear (138)

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 (143)

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 (111) 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 (135), or for arbitrary pointer data, TFPDataHashTable (117) is more suitable. The strings are exposed with their keys through the Items (145) property.

7.18.2 Method overview

Page	Property	Description
144	Add	Add a new string to the hash list

7.18.3 Property overview

Page	Property	Access	Description
145	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 (145)

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 (144)

7.19 THTCustomNode

7.19.1 Description

`THTCustomNode` is used by the TFPCustomHashTable (111) class to store the keys and associated values.

7.19.2 Method overview

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

7.19.3 Property overview

Page	Property	Access	Description
146	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 TFPHashTable (98) class when needed.

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

See also: TFPHashTable (98)

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 (146)

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 (145)

7.20 THTDataNode

7.20.1 Description

`THTDataNode` is used by TDataHashTable (98) 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`

7.20.2 Property overview

Page	Property	Access	Description
146	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 (145) descendent which holds the data in the TFPObjectHashTable (135) hash table. It exposes a data string.

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

7.21.2 Property overview

Page	Property	Access	Description
147	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 (136)

See also: TFPObjectHashTable (135), TFPObjectHashTable.Items (136), THTOwnedObjectNode (147)

7.22 THTOwnedObjectNode

7.22.1 Description

`THTOwnedObjectNode` is used instead of THTObjectNode (147) in case TFPObjectHashTable (135) owns it's objects. When this object is destroyed, the associated data object is also destroyed.

7.22.2 Method overview

Page	Property	Description
147	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 (147), TFPObjectHashTable.OwnsObjects (136)

7.23 THTStringNode

7.23.1 Description

`THTStringNode` is a THTCustomNode (145) descendent which holds the data in the TFPString-HashTable (144) hash table. It exposes a data string.

It should not be necessary to use `THTStringNode` directly, it's only for inner use by `TFPStringHashTable`

7.23.2 Property overview

Page	Property	Access	Description
148	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 (145)

See also: TFPStringHashTable (144)

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.

7.24.2 Method overview

Page	Property	Description
148	Add	Add an object to the list
149	Remove	Remove an object from the list

7.24.3 Property overview

Page	Property	Access	Description
149	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 (149), TObjectBucketList.Remove (149)

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 (148), TObjectBucketList.Data (149)

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 (148)

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 (137), `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.

7.25.2 Method overview

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

7.25.3 Property overview

Page	Property	Access	Description
153	Items	rw	Indexed access to the elements of the list.
152	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 (149), TObjectList.OwnsObjects (152), TFPObjectList (137)

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 (152), #rtl.classes.TList.Delete (**??**), TObjectList.Extract (150), TObjectList.Remove (151)

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 (151), `Extract` does not free the extracted element if OwnsObjects (152) 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 (151)

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 (152) 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 (150)

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 (151)

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 (151)

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 (150), TObjectList.Remove (151)

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 (152), TObjectList.Pack (149)

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 (152), TObjectList.Pack (149)

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 (150), TObjectList.Delete (149), TObjectList.Remove (151), TObjectList.Clear (149)

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
154	Peek	Look at the first object in the queue.
153	Pop	Pop the first element off the queue
153	Push	Push an object on the queue

7.26.2 TObjectQueue.Push

Synopsis: Push an object on the queue

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

Visibility: public

Description: `Push` pushes another object on the queue. It overrides the `Push` method as implemented in `TQueue` so it accepts only objects as arguments.

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

See also: TObjectQueue.Pop (153), TObjectQueue.Peek (154)

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 (153), TObjectQueue.Peek (154)

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 (153), TObjectQueue.Pop (153)

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.

7.27.2 Method overview

Page	Property	Description
155	Peek	Look at the top object in the stack.
154	Pop	Pop the top object of the stack.
154	Push	Push an object on the stack.

7.27.3 TObjectStack.Push

Synopsis: Push an object on the stack.

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

Visibility: public

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

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

See also: TObjectStack.Pop (154), TObjectStack.Peek (155)

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 (154), TObjectStack.Peek (155)

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 (154), TObjectStack.Pop (154)

7.28 TOrderedList

7.28.1 Description

`TOrderedList` provides the base class for TQueue (157) and TStack (157). 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.

7.28.2 Method overview

Page	Property	Description
156	AtLeast	Check whether the list contains a certain number of elements.
156	Count	Number of elements on the list.
155	Create	Create a new ordered list
155	Destroy	Free an ordered list
157	Peek	Return the next element to be popped from the list.
157	Pop	Remove an element from the list.
156	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 (155)

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 (155)

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 (156)

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 (156)

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 (157), TOrderedList.Peek (157)

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 (157), TOrderedList.Push (156)

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 (157), without actually popping it from the list.

Errors: None.

See also: TOrderedList.Pop (157), TOrderedList.Push (156)

7.29 TQueue

7.29.1 Description

`TQueue` is a descendent of TOrderedList (155) which implements Push (156) and Pop (157) 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 (155)

7.30 TStack

7.30.1 Description

`TStack` is a descendent of TOrderedList (155) which implements Push (156) and Pop (157) 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 (155)

Chapter 8

Reference for unit 'CustApp'

8.1 Used units

Table 8.1: Used units by unit 'CustApp'

Name	Page
Classes	??
sysutils	??

8.2 Overview

The CustApp unit implements the TCustomApplication (159) 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

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

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

8.3.2 Variables

```
CustomApplication : TCustomApplication = nil
```

`CustomApplication` contains the global application instance. All descendents of TCustomApplication (159) 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 whish to implement a global application class instance. It introduces several application-wide functionalities.

- Exception handling in HandleException (160), ShowException (161), OnException (166) and StopOnException (168).
- Command-line parameter parsing in FindOptionIndex (162), GetOptionValue (162), CheckOptions (163) and HasOption (163)
- Environment variable handling in GetEnvironmentList (164) and EnvironmentVariable (167).

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

8.4.2 Method overview

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

8.4.3 Property overview

Page	Property	Access	Description
168	CaseSensitiveOptions	rw	Are options interpreted case sensitive or not
166	ConsoleApplication	r	Is the application a console application or not
167	EnvironmentVariable	r	Environment variable access
164	ExeName	r	Name of the executable.
165	HelpFile	rw	Location of the application help file.
166	Location	r	Application location
166	OnException	rw	Exception handling event
167	OptionChar	rw	Command-line switch character
167	ParamCount	r	Number of command-line parameters
166	Params	r	Command-line parameters
168	StopOnException	rw	Should the program loop stop on an exception
165	Terminated	r	Was `Terminate` called or not
165	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 (160)

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 (160)

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 (166) 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 (161) 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 (168) property is set to `True`, the Terminated (165) property is set to `True`, which will cause the Run (161) loop to stop, and the application will exit.

See also: TCustomApplication.ShowException (161), TCustomApplication.StopOnException (168), TCustomApplication.Terminated (165), TCustomApplication.Run (161)

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 (161), TCustomApplication.Terminated (165)

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 (160). If TCustomApplication.StopOnException (168) 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 (160), TCustomApplication.StopException (159)

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: #rtl.sysutils.ShowException (**??**), TCustomApplication.HandleException (160), TCustomApplication.StopException (159)

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

See also: TCustomApplication.Terminated (165), TCustomApplication.Run (161)

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

Options are identified as command-line parameters which start with OptionChar (167) (by default the dash ('-') character).

See also: TCustomApplication.HasOption (163), TCustomApplication.GetOptionValue (162), TCustomApplication.CheckOptions (163), TCustomApplication.CaseSensitiveOptions (168), TCustomApplication.OptionChar (167)

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

Options are identified as command-line parameters which start with OptionChar (167) (by default the dash ('-') character).

See also: TCustomApplication.FindOptionIndex (162), TCustomApplication.HasOption (163), TCustomApplication.CheckOptions (163), TCustomApplication.CaseSensitiveOptions (168), TCustomApplication.OptionChar (167)

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

Options are identified as command-line parameters which start with OptionChar (167) (by default the dash ('-') character).

See also: TCustomApplication.FindOptionIndex (162), TCustomApplication.GetOptionValue (162), TCustomApplication.CheckOptions (163), TCustomApplication.CaseSensitiveOptions (168), TCustomApplication.OptionChar (167)

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) : String
function CheckOptions(const ShortOptions: String;
                      const Longopts: TStrings) : String
function CheckOptions(const ShortOptions: String;
                      const LongOpts: Array of String) : String
function CheckOptions(const ShortOptions: String;const LongOpts: String)
                       : String
```

Visibility: public

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

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

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

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

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

Options are identified as command-line parameters which start with OptionChar (167) (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: TCustomApplication.FindOptionIndex (162), TCustomApplication.GetOptionValue (162), TCustomApplication.HasOption (163), TCustomApplication.CaseSensitiveOptions (168), TCustomApplication.OptionChar (167)

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: TCustomApplication.EnvironmentVariable (167)

8.4.16 TCustomApplication.Log

Synopsis: Write a message to the event log

Declaration:
```
procedure Log(EventType: TEventType;Msg: String);  Virtual
```

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 (395)

Errors: None.

See also: #rtl.classes.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: #rtl.system.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 (165)

8.4.19 TCustomApplication.Terminated

Synopsis: Was `Terminate` called or not

Declaration: `Property Terminated : Boolean`

Visibility: public

Access: Read

Description: `Terminated` indicates whether Terminate (162) 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: TCustomApplication.Terminate (162)

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: TCustomApplication.HelpFile (165)

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 (159).

If set, `OnException` is called by the HandleEvent (159) routine. Do not use the `OnException` event directly, instead call `HandleEvent`

See also: TCustomApplication.ShowEvent (159)

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: #rtl.system.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: #rtl.system.paramstr (**??**), TCustomApplication.Params (166)

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 (167). It is equivalent to calling ParamStr (**??**).

See also: TCustomApplication.ParamCount (167), #rtl.system.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 (166) property.

See also: TCustomApplication.Params (166), #rtl.system.paramstr (**??**), #rtl.system.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 (164) can be used.

See also: TCustomApplication.GetEnvironmentList (164), TCustomApplication.Params (166)

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-alphanumerical character (although no check is performed on this).

See also: TCustomApplication.FindOptionIndex (162), TCustomApplication.GetOptionValue (162), TCustomApplication.HasOption (163), TCustomApplication.CaseSensitiveOptions (168), TCustomApplication.CheckOptions (163)

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 (162) and CheckOptions (163) 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: TCustomApplication.FindOptionIndex (162), TCustomApplication.GetOptionValue (162), TCustomApplication.HasOption (163), TCustomApplication.OptionChar (167), TCustomApplication.CheckOptions (163)

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 (161) and HandleException (160) procedures in case of an unhandled exception in the `DoRun` code. If `StopOnException` is `True` then Terminate (162) will be called after the exception was handled.

See also: TCustomApplication.Run (161), TCustomApplication.HandleException (160), TCustomApplication.Terminate (162)

Chapter 9

Reference for unit 'daemonapp'

9.1 Used units

Table 9.1: Used units by unit 'daemonapp'

Name	Page
Classes	??
CustApp	158
eventlog	394
rtlconsts	169
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 (184).

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 (189).

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 (192).

As can be seen, a single application can implement one ore more daemons (services). Each daemon will be run in a separate thread which is controlled by the application class.

The classes take care of logging through the TEventLog (395) 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 cre
```

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 = 'Options do not allow determining what needs to be done.'
```

No operation can be performed

```
SErrOnlyOneMapperAllowed = 'Not changing daemon mapper class %s with %s: Only 1 mapp
```

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

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 (192) definition.

```
TCustomDaemonMapperClass = Class of TCustomDaemonMapper
```

TCustomDaemonMapperClass is the class of TCustomDaemonMapper. It is used in the RegisterDaemonMapper (175) 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 (175) 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 succesfully.

```
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 (doAllowPause,doAllowStop,doInteractive)
```

TDaemonOption enumerates the various options a daemon can have.

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

Table 9.4: Enumeration values for type TDaemonRunMode

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

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

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

Table 9.5: Enumeration values for type TErrorSeverity

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

`TErrorSeverity` determines what action windows takes when the daemon fails to start. It is used on windows only, and is ignored on other platforms.

```
TGuiLoopEvent = procedure of object
```

`TGuiLoopEvent` is the main GUI loop event procedure prototype. It is called by the application instance in case the daemon has a visual part, which needs to handle visual events. It is run in the main application thread.

```
TServiceType = (stWin32,stDevice,stFileSystem)
```

Table 9.6: Enumeration values for type TServiceType

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

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

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

Table 9.7: Enumeration values for type TStartType

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

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

9.4.3 Variables

```
AppClass : TCustomDaemonApplicationClass
```

```
CurrentStatusNames : Array[TCurrentStatus] of String = ('Stopped','Start Pending','S
```

Names for various service statuses

```
DefaultDaemonOptions : TDaemonOptions = [doAllowStop,doAllowPause]
```

`DefaultDaemonOptions` are the default options with which a daemon definition (TDaemonDef (192)) is created.

```
SStatus : Array[1..5] of String = ('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 (178) instance used by this application. The instance is created at the first invocation of this function, so it is possible to use RegisterDaemonApplicationClass (174) to register an alternative `TCustomDaemonApplication` class to run the application.

See also: TCustomDaemonApplication (178), RegisterDaemonApplicationClass (174)

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: `DaemonError` raises an EDaemon (175) exception with message `Msg` or it formats the message using `Fmt` and `Args`.

See also: EDaemon (175)

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 (178) descendent which will be used when creating the global Application (174) instance. Only the last registered class pointer will be used.

See also: TCustomDaemonApplication (178), Application (174)

9.5.4 RegisterDaemonClass

Synopsis: Register daemon

Declaration: `procedure RegisterDaemonClass(AClass: TCustomDaemonClass)`

Visibility: default

Description: `RegisterDaemonClass` must be called for each TCustomDaemon (175) descendent that is used in the class: the class pointer and class name are used by the TCustomDaemonMapperClass (172) class to create a `TCustomDaemon` instance when a daemon is required.

See also: TCustomDaemonMapperClass (172), TCustomDaemon (175)

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 daemonmapper. The daemonmapper will be used only when the application is being run, by the TCustomDaemonApplication (178) code, so registering an alternative mapping class should happen in the initialization section of the application units.

See also: TCustomDaemonApplication (178), TCustomDaemonMapperClass (172)

9.6 EDaemon

9.6.1 Description

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

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

9.7.2 Method overview

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

9.7.3 Property overview

Page	Property	Access	Description
177	Controller	r	`TDaemonController` instance controlling this daemon instance
177	DaemonThread	r	Thread in which daemon is running
176	Definition	r	The definition used to instantiate this daemon instance
178	Logger	r	`TEventLog` instance used to send messages to the system log
177	Status	rw	Current status of the daemon

9.7.4 TCustomDaemon.LogMessage

Synopsis: Log a message to the system log

Declaration: `procedure LogMessage(Msg: String)`

Visibility: public

Description: `LogMessage` can be used to send a message `Msg` to the system log. A TEventLog (395) instance is used to actually send messages to the system log.

The message is sent with an 'error' flag (using TEventLog.Error (398)).

Errors: None.

See also: TCustomDaemon.ReportStatus (176)

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: TCustomDaemon.LogMessage (176)

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 (192) 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 (192)

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: TCustomDaemon.Controller (177)

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: TCustomDaemon.DaemonThread (177)

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: TCustomDaemon.ReportStatus (176)

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 (395) instance used to send messages to the system log. It is used by the LogMessage (176) call, but is acessible through the `Logger` property in case more configurable logging is needed than offered by `LogMessage`.

See also: TCustomDaemon.LogMessage (176), #fcl.eventlog.TEventLog (395)

9.8 TCustomDaemonApplication

9.8.1 Description

`TCustomDaemonApplication` is a TCustomApplication (159) 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 (184) instances by checking the TCustomDaemonMapperClass (172) instance that contains the daemon maps.

9.8.2 Method overview

Page	Property	Description
179	CreateDaemon	Create daemon instance
180	CreateForm	Create a component
179	InstallDaemons	Install all daemons.
179	RunDaemons	Run all daemons.
178	ShowException	Show an exception
179	StopDaemons	Stop all daemons
180	UnInstallDaemons	Uninstall all daemons

9.8.3 Property overview

Page	Property	Access	Description
181	GuiHandle	rw	Handle of GUI loop main application window handle
181	GUIMainLoop	rw	GUI main loop callback
180	Logger	r	Event logging instance used for logging messages
180	OnRun	rw	
181	RunMode	r	Application mode

9.8.4 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.

9.8.5 TCustomDaemonApplication.CreateDaemon

Synopsis: Create daemon instance

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

Visibility: public

Description: `CreateDaemon` is called whenever a TCustomDaemon (175) instance must be created from a TDaemonDef (192) daemon definition, passed in `DaemonDef`. It initializes the `TCustomDaemon` instance, and creates a controller instance of type TDaemonController (189) to control the daemon. Finally, it assigns the created daemon to the TDaemonDef.Instance (193) property.

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

See also: TDaemonController (189), TCustomDaemon (175), TDaemonDef (192), TDaemonDef.Instance (193)

9.8.6 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 (190), TCustomDaemonApplication.UnInstallDaemons (180), TCustomDaemonApplication.RunDaemons (179)

9.8.7 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 (180), TCustomDaemonApplication.RunDaemons (179), TCustomDaemonApplication.StopDaemons (179)

9.8.8 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.

Errors:

See also: TCustomDaemonApplication.UnInstallDaemons (180), TCustomDaemonApplication.InstallDaemons (179), TCustomDaemonApplication.StopDaemons (179)

9.8.9 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 (179), TCustomDaemonApplication.InstallDaemons (179), TCustomDaemonApplication.StopDaemons (179)

9.8.10 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 (179)

9.8.11 TCustomDaemonApplication.OnRun

Declaration: `Property OnRun : TNotifyEvent`

Visibility: public

Access: Read,Write

9.8.12 TCustomDaemonApplication.Logger

Synopsis: Event logging instance used for logging messages

Declaration: `Property Logger : TEventLog`

Visibility: public

Access: Read

Description: `Logger` contains a reference to the TEventLog (395) instance that can be used to send messages to the system log.

See also: TCustomDaemon.LogMessage (176)

9.8.13 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 (181)

9.8.14 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 (181) procedure exists, and the application creates and runs a message loop by itself.

See also: TCustomDaemonApplication.GUIMainLoop (181)

9.8.15 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 (179), RunDaemons (179) or UnInstallDaemons (180) is called.

See also: TCustomDaemonApplication.InstallDaemons (179), TCustomDaemonApplication.RunDaemons (179), TCustomDaemonApplication.UnInstallDaemons (180)

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 (196) 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 (192) definitions and the actual TDaemon (184) instances that will be used to run the daemons.

The `TCustomDaemonMapper` class has no support for streaming. The TDaemonMapper (198) class has support for streaming (and hence visual designing).

9.9.2 Method overview

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

9.9.3 Property overview

Page	Property	Access	Description
183	DaemonDefs	rw	Collection of daemons
183	OnCreate	rw	Event called when the daemon mapper is created
183	OnDestroy	rw	Event called when the daemon mapper is freed.
184	OnInstall	rw	Event called when the daemons are installed
183	OnRun	rw	Event called when the daemons are executed.
184	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 (196) 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 (196), TCustomDaemonApplication (178), TCustomDaemonMapper.Destroy (182)

9.9.5 TCustomDaemonMapper.Destroy

Synopsis: Clean up and destroy a `TCustomDaemonMapper` instance.

Declaration: `destructor Destroy;  Override`

Visibility: public

Description: `Destroy` frees the DaemonDefs (183) collection and calls the inherited destructor.

See also: TDaemonDefs (196), TCustomDaemonMapper.Create (182)

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 (178)

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: TCustomDaemonMapper.OnDestroy (183), TCustomDaemonMapper.OnUnInstall (184), TCustomDaemonMapper.OnCreate (183), TCustomDaemonMapper.OnDestroy (183)

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 (183) event.

See also: TCustomDaemonMapper.OnCreate (183), TCustomDaemonMapper.OnInstall (184), TCustomDaemonMapper.OnUnInstall (184), TCustomDaemonMapper.OnCreate (183)

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: TCustomDaemonMapper.OnInstall (184), TCustomDaemonMapper.OnUnInstall (184), TCustomDaemonMapper.OnCreate (183), TCustomDaemonMapper.OnDestroy (183)

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: TCustomDaemonMapper.OnRun (183), TCustomDaemonMapper.OnUnInstall (184), TCustomDaemonMapper.OnCreate (183), TCustomDaemonMapper.OnDestroy (183)

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: TCustomDaemonMapper.OnRun (183), TCustomDaemonMapper.OnInstall (184), TCustomDaemonMapper.OnCreate (183), TCustomDaemonMapper.OnDestroy (183)

9.10 TDaemon

9.10.1 Description

`TDaemon` is a TCustomDaemon (175) 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 (177)), which is not the main program thread.

9.10.2 Property overview

Page	Property	Access	Description
188	AfterInstall	rw	Called after the daemon was installed
188	AfterUnInstall	rw	Called after the daemon is uninstalled
187	BeforeInstall	rw	Called before the daemon will be installed
188	BeforeUnInstall	rw	Called before the daemon is uninstalled
185	Definition		
186	OnContinue	rw	Daemon continue
188	OnControlCode	rw	Called when a control code is received for the daemon
187	OnExecute	rw	Daemon execute event
186	OnPause	rw	Daemon pause event
187	OnShutDown	rw	Daemon shutdown
185	OnStart	rw	Daemon start event
186	OnStop	rw	Daemon stop event
185	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 (176) 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 (187) event handler, so the event handler can return at once.

See also: TDaemon.OnStop (186), TDaemon.OnExecute (187), TDaemon.OnContinue (186), TCustomDaemon.ReportStatus (176)

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 (176) method.

If a thread was started in the OnExecute (187) event, this is the place where the thread should be stopped.

See also: TDaemon.OnStart (185), TDaemon.OnPause (186), TCustomDaemon.ReportStatus (176)

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 (176) method.

If a thread was started in the OnExecute (187) event, this is the place where the thread's execution should be suspended.

See also: TDaemon.OnStop (186), TDaemon.OnContinue (186), TCustomDaemon.ReportStatus (176)

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 (176) method.

If a thread was started in the OnExecute (187) event and it was suspended in a OnPause (185) event, this is the place where the thread's executed should be resumed.

See also: TDaemon.OnStart (185), TDaemon.OnPause (186), TCustomDaemon.ReportStatus (176)

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

See also: TDaemon.OnStart (185), TDaemon.OnPause (186), TCustomDaemon.ReportStatus (176)

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 (185), TDaemon.OnStop (186)

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: TDaemon.AfterInstall (188), TDaemon.BeforeUnInstall (188), TDaemon.AfterUnInstall (188)

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: TDaemon.BeforeInstall (187), TDaemon.BeforeUnInstall (188), TDaemon.AfterUnInstall (188)

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: TDaemon.BeforeInstall (187), TDaemon.AfterInstall (188), TDaemon.AfterUnInstall (188)

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: TDaemon.BeforeInstall (187), TDaemon.AfterInstall (188), TDaemon.BeforeUnInstall (188)

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: daemonapp.architecture (169)

9.11 TDaemonApplication

9.11.1 Description

`TDaemonApplication` is the default TCustomDaemonApplication (178) descendent that is used to run the daemon application. It is possible to register an alternative `TCustomDaemonApplication` class (using RegisterDaemonApplicationClass (174)) to run the application in a different manner.

9.12 TDaemonController

9.12.1 Description

`TDaemonController` is a class that is used by the TDaemonApplication (189) 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.

9.12.2 Method overview

Page	Property	Description
190	Controller	Controller
189	Create	Create a new instance of the `TDaemonController` class
190	Destroy	Free a `TDaemonController` instance.
190	Main	Daemon main entry point
191	ReportStatus	Report the status to the operating system.
190	StartService	Start the service

9.12.3 Property overview

Page	Property	Access	Description
192	CheckPoint		Send checkpoint signal to the operating system
191	Daemon	r	Daemon instance this controller controls.
191	LastStatus	r	Last reported status
191	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 (189) instance, and `AOwner` will be set to the global TDaemonApplication (189) instance.

See also: TDaemonApplication (189), TDaemonController.Destroy (190)

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: TDaemonController.Create (189)

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 (190)

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 (198)

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 (188)

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 (184), TDaemonController.LastStatus (191)

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 (190) 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: TDaemonController.ReportStatus (191)

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: TDaemonController.ReportStatus (191)

9.13 TDaemonDef

9.13.1 Description

`TDaemonDef` contains the definition of a daemon in the application: The name of the daemon, which TCustomDaemon (175) 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.

9.13.2 Method overview

Page	Property	Description
192	Create	Create a new `TDaemonDef` instance
193	Destroy	Free a `TDaemonDef` from memory

9.13.3 Property overview

Page	Property	Access	Description
193	DaemonClass	r	`TDaemon` class to use for this daemon
193	DaemonClassName	rw	Name of the `TDaemon` class to use for this daemon
194	Description	rw	Description of the daemon
194	DisplayName	rw	Displayed name of the daemon (service)
195	Enabled	rw	Is the daemon enabled or not
193	Instance	rw	Instance of the daemon class
196	LogStatusReport	rw	Log the status report to the system log
194	Name	rw	Name of the daemon (service)
195	OnCreateInstance	rw	Event called when a daemon in instantiated
195	Options	rw	Service options
194	RunArguments	rw	Additional command-line arguments when running daemon.
195	WinBindings	rw	Windows-specific bindings (windows only)

9.13.4 TDaemonDef.Create

Synopsis: Create a new `TDaemonDef` instance

Declaration: `constructor Create(ACollection: TCollection); Override`

Visibility: public

Description: `Create` initializes a new `TDaemonDef` instance. It should not be necessary to instantiate a definition manually, it is handled by the collection.

See also: TDaemonDefs (196)

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 (193).

See also: TDaemonDef.DaemonClassName (193), TDaemonMapper (198)

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 (184) instance that is used when the service is in operation at runtime.

See also: TDaemonDef.DaemonClass (193)

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 (178) creates an instance of the daemon.

See also: TDaemonDef.Instance (193), TDaemonDef.DaemonClass (193), RegisterDaemonClass (175)

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 (194)

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 (194)

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 (203)

9.13.16 TDaemonDef.OnCreateInstance

Synopsis: Event called when a daemon in instantiated

Declaration: `Property OnCreateInstance : TNotifyEvent`

Visibility: published

Access: Read,Write

Description: `OnCreateInstance` is called whenever an instance of the daemon is created. This can be used for instance when a single `TDaemon` class is used to run several services, to correctly initialize the TDaemon.

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 (184)

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 (182) instance will create a collection and maintain it.

9.14.2 Method overview

Page	Property	Description
196	Create	Create a new instance of a `TDaemonDefs` collection.
197	DaemonDefByName	Find and return instance of daemon definition with given name.
197	FindDaemonDef	Find and return instance of daemon definition with given name.
197	IndexOfDaemonDef	Return index of daemon definition

9.14.3 Property overview

Page	Property	Access	Description
197	Daemons	rw	Indexed access to `TDaemonDef` instances

9.14.4 TDaemonDefs.Create

Synopsis: Create a new instance of a `TDaemonDefs` collection.

Declaration: `constructor Create(AOwner: TPersistent;AClass: TCollectionItemClass)`

Visibility: public

Description: `Create` creates a new instance of the `TDaemonDefs` collection. It keeps the `AOwner` parameter for future reference and calls the inherited constructor.

Normally it is not necessary to create an instance of `TDaemonDefs` manually. The global TCustomDaemonMapper (182) instance will create a collection and maintain it.

See also: TDaemonDef (192)

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

See also: TDaemonDefs.FindDaemonDef (197), TDaemonDefs.DaemonDefByName (197)

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

See also: TDaemonDefs.IndexOfDaemonDef (197), TDaemonDefs.DaemonDefByName (197)

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 (192) instance whose name matches `DaemonName`. If no definition is found, an EDaemon (175) exception is raised.

The FindDaemonDef (197) call does not raise an error, but returns `Nil` instead.

Errors: If no definition is found, an EDaemon (175) exception is raised.

See also: TDaemonDefs.IndexOfDaemonDef (197), TDaemonDefs.FindDaemonDef (197)

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 (192)

9.15 TDaemonMapper

9.15.1 Description

`TDaemonMapper` is a direct descendent of TCustomDaemonMapper (182), but introduces no new functionality. It's sole purpose is to make it possible for an IDE to stream the `TDaemonMapper` instance.

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

If the instance should not be streamed, the CreateNew (198) constructor can be used instead.

9.15.2 Method overview

Page	Property	Description
198	Create	Create a new `TDaemonMapper` instance and initializes it from streamed resources.
198	CreateNew	Create a new `TDaemonMapper` instance without initialization

9.15.3 TDaemonMapper.Create

Synopsis: Create a new `TDaemonMapper` instance and initializes it from streamed resources.

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

Visibility: default

Description: `Create` initializes a new instance of `TDaemonMapper` and attempts to read the component from resources compiled in the application.

If the instance should not be streamed, the CreateNew (198) 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: TDaemonMapper.CreateNew (198)

9.15.4 TDaemonMapper.CreateNew

Synopsis: Create a new `TDaemonMapper` instance without initialization

Declaration: `constructor CreateNew(AOwner: TComponent;Dummy: Integer)`

Visibility: default

Description: `CreateNew` itializes a new instance of `TDaemonMapper`. In difference with the `Create` constructor, it does not attempt to read the component from a stream.

See also: TDaemonMapper.Create (198)

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 (189) class will take care of this.

9.16.2 Method overview

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

9.16.3 Property overview

Page	Property	Access	Description
201	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 (201)

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 (184)

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 (184) method, and terminates the thread.

See also: TDaemon.Stop (184), TDaemonThread.PauseDaemon (200), TDaemonThread.ShutDownDaemon (200)

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

See also: TDaemon.Pause (184), TDaemonThread.StopDaemon (200), TDaemonThread.ContinueDaemon (200), TDaemonThread.ShutDownDaemon (200)

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 (184) method. It returns `True` if the attempt was succesful.

See also: TDaemon.Continue (184), TDaemonThread.StopDaemon (200), TDaemonThread.PauseDaemon (200), TDaemonThread.ShutDownDaemon (200)

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 (184) function. The thread is terminated by this method.

See also: TDaemon.Shutdown (184), TDaemonThread.StopDaemon (200), TDaemonThread.PauseDaemon (200), TDaemonThread.ContinueDaemon (200)

9.16.11 TDaemonThread.InterrogateDaemon

Synopsis: Report the daemon status

Declaration: `function InterrogateDaemon : Boolean;  Virtual`

Visibility: public

Description: `InterrogateDaemon` simply calls TDaemon.ReportStatus (184) for the daemon that is running in this thread. It always returns `True`.

See also: TDaemon.ReportStatus (184)

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 (184)

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 (202) instances.

9.17.2 Method overview

Page	Property	Description
201	Create	Create a new instance of a `TDependencies` collection.

9.17.3 Property overview

Page	Property	Access	Description
202	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 (202).

See also: TDependency (202)

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.

9.18.2 Method overview

Page	Property	Description
202	Assign	Assign `TDependency` instance to another

9.18.3 Property overview

Page	Property	Access	Description
203	IsGroup	rw	Name refers to a service group
202	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 (203)

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 (202)

9.19 TWinBindings

9.19.1 Description

`TWinBindings` contains windows-specific properties for the daemon definition (in TDaemonDef.WinBindings (195)). If the daemon should not run on Windows, then the properties can be ignored.

9.19.2 Method overview

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

9.19.3 Property overview

Page	Property	Access	Description
205	Dependencies	rw	Service dependencies
204	ErrCode	rw	Service specific error code
207	ErrorSeverity	rw	Error severity in case of startup failure
205	GroupName	rw	Service group name
206	IDTag	rw	Location in the service group
205	Password	rw	Password for service startup
207	ServiceType	rw	Type of service
206	StartType	rw	Service startup type.
205	UserName	rw	Username to run service as
206	WaitHint	rw	Timeout wait hint
204	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 (192), TDaemonDef.WinBindings (195), TWinBindings.Dependencies (205)

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 (205), TWinBindings.Create (203)

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 (184) to the windows service manager. If it is zero, then the contents of Win32ErrCode (204) are reported. If it is nonzero, then the windows-errorcode is set to `ERROR_SERVICE_SPECIFIC_ERROR`.

See also: TWinBindings.Win32ErrCode (204)

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 (184) to the windows service manager. It is sent if ErrCode (204) is zero.

See also: TWinBindings.ErrCode (204)

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 (203)

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: TWinBindings.UserName (205)

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 (205) property.

This property is only used during installation of the service.

See also: TWinBindings.Password (205)

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 (184)

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	208
sysutils	??
Variants	208

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 (250) representing a set of records from a database, TField (298) which represents the contents of a field in a record, TDatasource (287) which acts as an event distributor on behalf of a dataset and TParams (368) which can be used to parametrize queries. The databases connections themselves are abstracted in the TDatabase (241) class.

10.3 Constants, types and variables

10.3.1 Constants

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

`DefaultFieldClasses` contains the TField (298) descendent class to use when a `TDataset` instance needs to create fields based on the TFieldDefs (326) 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 (276) 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 div8
```

Maximum data buffers count for dataset

```
dsMaxStringSize = 8192
```

Maximum size of string fields

```
dsWriteModes = [dsEdit,dsInsert,dsSetKey,dsCalcFields,dsFilter,dsNewValue,dsInternal
```

dsWriteModes contains the various values of TDataset.State (276) for which data can be written to the dataset buffer.

```
Fieldtypenames : Array[TFieldType] of String = ('Unknown','String','Smallint','Integ
```

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

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

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 (250) or one of it's descendents.

```
PBufferList = ^TBufferList
```

PBufferList is a pointer to a structure of type TBufferList (211). 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 = String
```

`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 (250) method. It is an opaque type, and should not be used any more, it is superseded by the TBookmarkStr (211) 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 (250) and it's descendent types to mark the internal memory buffers. It should not be used in end-user applications.

```
TBookmarkStr = String
```

TBookmarkStr is the type used by the TDataset.Bookmark (271) property. It can be used as a string, but should in fact be considered an opaque type.

```
TBufferArray = ^pchar
```

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 PChar
```

TBufferList is used intenally by the TDataset (250) 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
daAbort	The operation should be aborted (edits are undone, and an EAbort exception is raised)
daFail	The operation should fail (an exception will be raised)
daRetry	Retry the operation.

TDataAction is used by the TDataSetErrorEvent (212) 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 (241) class.

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

TDataChangeEvent is the event handler prototype for the TDatasource.OnDataChange (290) 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)
```

TDataEvent describes the various events that can be sent to TDatasource (287) instances connected to a TDataset (250) instance.

```
TDataOperation = procedure of object
```

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

`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 (250) class. It is curently 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 (286), TDataset.OnPostError (287) and TDataset.OnDeleteError (285) event handlers to allow the programmer to specfy what should be done if an update operation fails with an exception: The `Dataset` parameter indicates what dataset triggered the event, the `E` parameter contains the exception object. The `DataAction` must be set by the event handler, and based on it's return value, the dataset instance will take appropriate action. The default value is `daFail`, i.e. the exception will be raised again. For a list of available return values, see TDataAction (211).

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

`TDatasetNotifyEVent` is used in most of the TDataset (250) 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)
```

`TDataSetState` describes the current state of the dataset. During it's lifetime, the dataset's state is described by these enumerated values.

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.

Some state are not used in the default TDataset implementation, and are only used by certain descendents.

```
TDateTimeAlias = TDateTime
```

`TDateTimeAlias` is no longer used.

```
TDateTimeRec = record
end
```

`TDateTimeRec` was used by older TDataset (250) 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 (293)

```
TDBTransactionClass = Class of TDBTransaction
```

`TDBTransactionClass` is the class pointer for the TDBTransaction (294) class.

```
TFieldAttribute = (faHiddenCol,faReadonly,faRequired,faLink,faUnNamed,
                   faFixed)
```

`TFieldAttribute` is used to denote some attributes of a field in a database. It is used in the Attributes (325) property of TFieldDef (322).

```
TFieldAttributes= Set of (faFixed,faHiddenCol,faLink,faReadonly,
                          faRequired,faUnNamed)
```

`TFieldAttributes` is used in the TFieldDef.Attributes (325) property to denote additional attributes of the underlying field.

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

```
TFieldChars = Set of Char
```

`TFieldChars` is a type used in the TField.ValidChars (313) 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 (321) 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 (fkCalculated,fkData,fkInternalCalc,fkLookup)
```

`TFieldKinds` is a set of TFieldKind (214) 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 (298) 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)
```

TFieldType indicates the type of a TField (298) underlying data, in the DataType (310) property.

```
TFilterOption = (foCaseInsensitive,foNoPartialCompare)
```

TFilterOption enumerates the various options available when filtering a dataset. The TFilterOptions (215) set is used in the TDataset.FilterOptions (278) property to indicate which of the options should be used when filtering the data.

```
TFilterOptions= Set of (foCaseInsensitive,foNoPartialCompare)
```

TFilterOption is the set of filter options to use when filtering a dataset. This set type is used in the TDataset.FilterOptions (278) property. The available values are described in the TFilterOption (215) type.

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

TFilterRecordEvent is the prototype for the TDataset.OnFilterRecord (286) 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)
```

TGetMode is used internally by TDataset (250) 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)
```

TGetResult is used by descendents of TDataset (250) when they have to communicate the result of the GetRecord operation back to the TDataset record.

```
TIndexOption = (ixPrimary,ixUnique,ixDescending,ixCaseInsensitive,
                ixExpression,ixNonMaintained)
```

TIndexOption describes the various properties that an index can have. It is used in the TIndexOptions (216) set type to describe all properties of an index definition as in TIndexDef (338).

```
TIndexOptions= Set of (ixCaseInsensitive,ixDescending,ixExpression,
                       ixNonMaintained,ixPrimary,ixUnique)
```

TIndexOptions contains the set of properties that an index can have. It is used in the TIndexDef.Options (340) property to describe all properties of an index definition as in TIndexDef (338).

```
TIntegerField = TLongintField
```

TIntegerField is an alias for TLongintField (346).

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

TLocateOption is used in the TDataset.Locate (266) call to enumerate the possible options available when locating a record in the dataset.

```
TLocateOptions= Set of (loCaseInsensitive,loPartialKey)
```

TLocateOptions is used in the TDataset.Locate (266) 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 (240) 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 axuiliary type used when parsing and binding parameters in SQL statements. It should never be used directly in application code.

```
TParamStyle = (psInterbase,psPostgreSQL,psSimulated)
```

TParamStyle denotes the style in which parameters are specified in a query. It is used in the TParams.ParseSQl (370) 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)
```

TParamType indicates the kind of parameter represented by a TParam (356) instance. it has one of the following values:

ptUnknown Unknown type

ptInput Input parameter

ptOutput Output paramete, filled on result

ptInputOutput Input/output parameter

ptResult Result parameter

```
TParamTypes= Set of (ptInput,ptInputOutput,ptOutput,ptResult,ptUnknown)
```

TParamTypes is defined for completeness: a set of TParamType (217) values.

```
TProviderFlag = (pfInUpdate,pfInWhere,pfInKey,pfHidden)
```

TProviderFlag describes how the field should be used when applying updates from a dataset to the database. Each field of a TDataset (250) has one or more of these flags.

```
TProviderFlags= Set of (pfHidden,pfInKey,pfInUpdate,pfInWhere)
```

TProviderFlags is used for the TField.ProviderFlags (319) property to describe the role of the field when applying updates to a database.

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

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)
```

TResyncMode is used internally by various TDataset (250) navigation and data manipulation methods such as the TDataset.Refresh (269) 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)
```

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 (250) 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)
```

TUpdateKind indicates what kind of update operation is in progress when applying updates.

```
TUpdateMode = (upWhereAll,upWhereChanged,upWhereKeyOnly)
```

TUpdateMode determines how the WHERE clause of update queries for SQL databases should be constructed.

```
TUpdateStatus = (usUnmodified,usModified,usInserted,usDeleted)
```

TUpdateStatus determines the current state of the record buffer, if updates have not yet been applied to the database.

```
TUpdateStatusSet= Set of (usDeleted,usInserted,usModified,usUnmodified)
```

TUpdateStatusSet is a set of TUpdateStatus (218) 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.system.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 (220) exception, passing it Msg. If Comp is specified, the name of the component is prepended to the message.

See also: DatabaseErrorFmt (219), EDatabaseError (220)

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 (220) exception, passing it a message made by calling rtl.sysutils.format (208) with the `fmt` and `Args` arguments. If `Comp` is specified, the name of the component is prepended to the message.

See also: DatabaseError (218), EDatabaseError (220)

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 (215), TDateTimeRec (213), DateTimeToDateTimeRec (219)

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 valuein `Data` to a `TDateTimeRec` record.

Errors:

See also: TFieldType (215), TDateTimeRec (213), DateTimeRecToDateTime (219)

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 (329)

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 (370)

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 (218) call.

10.6 EUpdateError

10.6.1 Description

`EupdateError` is an exception used by the TProvider database support. It should never be raised directly.

10.6.2 Method overview

Page	Property	Description
221	Create	Create a new EUpdateError instance
221	Destroy	Free the EupdateError instance

10.6.3 Property overview

Page	Property	Access	Description
221	Context	r	Context in which exception occurred.
222	ErrorCode	r	Numerical error code.
222	OriginalExcaption	r	Original exception object, if available.
222	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 (220)

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 (220)

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 (220), EUpdateError.ErrorCode (222), EUpdateError.PreviousError (222)

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 (220), EUpdateError.Context (221), EUpdateError.PreviousError (222)

10.6.8 EUpdateError.OriginalExcaption

Synopsis: Original exception object, if available.

Declaration: `Property OriginalExcaption : Exception`

Visibility: public

Access: Read

Description: `OriginalException` is the original exception object as raised by the database access layer. It may or may not be available.

See also: EUpdateError.ErrorCode (222), EUpdateError.Context (221), EUpdateError.PreviousError (222)

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 (222), EUpdateError.Context (221), EUpdateError.OriginalException (220)

10.7 TAutoIncField

10.7.1 Description

`TAutoIncField` is the class created when a dataset must manage 32-bit signed integer data, of datatype `ftAutoInc`: This field gets it's data automatically by the database engine. It exposes no new properties, but simply overrides some methods to manage 32-bit signed integer data.

It should never be necessary to create an instance of `TAutoIncField` manually, a field of this class will be instantiated automatically for each auto-incremental field when a dataset is opened.

10.7.2 Method overview

Page	Property	Description
223	Create	Create a new instance of the `TAutoIncField` class.

10.7.3 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 (298) class' fields.

See also: TField (298)

10.8 TBCDField

10.8.1 Description

`TBCDField` is the class used when a dataset must manage data of Binary Coded Decimal type. (TField.DataType (310) equals `ftBCD`). It initializes some of the properties of the TField (298) 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 (208) 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.

10.8.2 Method overview

Page	Property	Description
224	CheckRange	Check whether a values falls within the allowed range
223	Create	Create a new instance of a `TBCDField` class.

10.8.3 Property overview

Page	Property	Access	Description
225	Currency	rw	Does the field represent a currency amount
225	MaxValue	rw	Maximum value for the field
225	MinValue	rw	Minimum value for the field
224	Precision	rw	Precision of the BCD field
226	Size		Number of decimals after the decimal separator
224	Value	rw	Value of the field contents as a Currency type

10.8.4 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 (298) properties to configure the instance for working with BCD data values.

See also: TField (298)

10.8.5 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 (225) and MaxValue (225) properties. If the value lies outside of the allowed range, then `False` is returned.

See also: TBCDField.MaxValue (225), TBCDField.MinValue (225)

10.8.6 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 (314) property to a currency type field. It returns the same value as the TField.AsCurrency (306) field.

See also: TField.Value (314), TField.AsCurrency (306)

10.8.7 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 (226), 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 (226), TBCDField.Value (224)

10.8.8 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 (311) and TField.Text (313) 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 (311), TField.Text (313)

10.8.9 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, MinField (208) should also be set, because it will also be checked.

See also: TBCDField.MinValue (225), TBCDField.CheckRange (224)

10.8.10 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, MaxField (208) should also be set, because it will also be checked.

See also: TBCDField.MaxValue (225), TBCDField.CheckRange (224)

10.8.11 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 (224) field.

See also: TBCDField.Precision (224)

10.9 TBinaryField

10.9.1 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 (298) 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 (232) or TVarBytesField (375) should be created.

10.9.2 Method overview

Page	Property	Description
226	Create	Create a new instance of a `TBinaryField` class.

10.9.3 Property overview

Page	Property	Access	Description
226	Size		Size of the binary data

10.9.4 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 (298)

10.9.5 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 (313)

10.10 TBlobField

10.10.1 Description

`TBLobField` is the class used when a dataset must manage BLOB data. (TField.DataType (310) equals `ftBLOB`). It initializes some of the properties of the TField (298) 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 (353), TWideMemoField (377) or TGraphicField (337)

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.

10.10.2 Method overview

Page	Property	Description
228	Clear	Clear the BLOB field's contents
227	Create	Create a new instance of a `TBlobField` class.
228	IsBlob	Is the field a blob field
228	LoadFromFile	Load the contents of the field frop a file
228	LoadFromStream	Load the field's contents from stream
229	SaveToFile	Save field contents to a file
229	SaveToStream	Save the field's contents to stream
229	SetFieldType	Set field type

10.10.3 Property overview

Page	Property	Access	Description
229	BlobSize	r	Size of the current blob
230	BlobType	rw	Type of blob
230	Modified	rw	Has the field's contents been modified.
231	Size		Size of the blob field
230	Transliterate	rw	Should the contents of the field be transliterated
230	Value	rw	Return the field's contents as a string

10.10.4 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 (298) properties to configure the instance for working with BLOB data.

See also: TField (298)

10.10.5 TBlobField.Clear

Synopsis: Clear the BLOB field's contents

Declaration: `procedure Clear;  Override`

Visibility: public

Description: `Clear` overrides the `TField` implementation of TField.Clear (303). It creates and immediatly releases an empty blob stream in write mode, effectively clearing the contents of the BLOB field.

See also: TField.Clear (303), TField.IsNull (312)

10.10.6 TBlobField.IsBlob

Synopsis: Is the field a blob field

Declaration: `function IsBlob : Boolean;  Override`

Visibility: public

Description: `IsBlob` is overridden by `TBlobField` to return `True`

See also: TField.IsBlob (304)

10.10.7 TBlobField.LoadFromFile

Synopsis: Load the contents of the field frop 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 (228) 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: TBlobField.LoadFromStream (228), TBlobField.SaveToFile (229)

10.10.8 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: TBlobField.SaveToStream (229), TBlobField.LoadFromFile (228)

10.10.9 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 (229) 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: TBlobField.LoadFromFile (228), TBlobField.SaveToStream (229)

10.10.10 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: TBlobField.SaveToFile (229), TBlobField.LoadFromStream (228)

10.10.11 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 (310)

10.10.12 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 (313), TField.DataSize (310)

10.10.13 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 (228)

10.10.14 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 (308) property.

See also: TField.Value (314), TField.AsString (308)

10.10.15 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 (270) 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 (374), TDataset.Translate (270)

10.10.16 TBlobField.BlobType

Synopsis: Type of blob

Declaration: `Property BlobType : TBlobType`

Visibility: published

Access: Read,Write

Description: `BlobType` is an alias for TField.DataType (310), but with a restricted set of values. Setting `BlobType` is equivalent to setting the TField.DataType (310) property.

See also: TField.DataType (310)

10.10.17 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 (229) property instead.

See also: TBlobField.BlobSize (229)

10.11 TBooleanField

10.11.1 Description

`TBooleanField` is the field class used by TDataset (250) whenever it needs to manage boolean data (TField.DataType (310) equals `ftBoolean`). It overrides some properties and methods of TField (298) 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.

10.11.2 Method overview

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

10.11.3 Property overview

Page	Property	Access	Description
232	DisplayValues	rw	Textual representation of the true and false values
232	Value	rw	Value of the field as a boolean value

10.11.4 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 (298) properties to configure it for working with boolean values.

See also: TField (298)

10.11.5 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 (314) by `TBooleanField` as a boolean value. It returns the same value as the TField.AsBoolean (306) property.

See also: TField.AsBoolean (306), TField.Value (314)

10.11.6 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 (308), TField.Text (313)

10.12 TBytesField

10.12.1 Description

`TBytesField` is the class used when a dataset must manage data of fixed-size binary type. (TField.DataType (310) equals `ftBytes`). It initializes some of the properties of the TField (298) 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.

10.12.2 Method overview

Page	Property	Description
233	Create	Create a new instance of a `TBytesField` class.

10.12.3 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 (298) properties to configure the instance for working with binary data values.

See also: TField (298)

10.13 TCheckConstraint

10.13.1 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 (234) property , or a constraint enforced by the user on a particular dataset instance stored in TCheckConstraint.CustomConstraint (234)

10.13.2 Method overview

Page	Property	Description
233	Assign	Assign one constraint to another

10.13.3 Property overview

Page	Property	Access	Description
234	CustomConstraint	rw	User-defined constraint
234	ErrorMessage	rw	Message to display when the constraint is violated
234	FromDictionary	rw	True if the constraint is imported from a datadictionary
234	ImportedConstraint	rw	Constraint imported from the database engine

10.13.4 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 (234), TCheckConstraint.CustomConstraint (234)

10.13.5 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 (250) 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 TCustomConstraint.ErrorMessage (208)

See also: TCustomConstraint.ErrorMessage (208)

10.13.6 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 (234)

10.13.7 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 (250) descendents to indicate the source of the constraint, but is otherwise ignored.

See also: TCheckConstraint.ImportedConstraint (234)

10.13.8 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 (250) descendent.

See also: TCheckConstraint.CustomConstraint (234)

10.14 TCheckConstraints

10.14.1 Description

`TCheckConstraints` is a `TCollection` descendent which keeps a collection of TCheckConstraint (233) items. It overrides the Add (235) method to return a `TCheckConstraint` instance.

10.14.2 Method overview

Page	Property	Description
235	Add	Add new `TCheckConstraint` item to the collection
235	Create	Create a new instance of the `TCheckConstraints` class.

10.14.3 Property overview

Page	Property	Access	Description
235	Items	rw	Indexed access to the items in the collection

10.14.4 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 (250) 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 (233) class pointer.

See also: TCheckConstraint (233), TDataset (250)

10.14.5 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 (233) instance to the collection. it returns the newly added instance.

See also: TCheckConstraint (233), #rtl.classes.TCollection.Add (**??**)

10.14.6 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 (**??**)

10.15 TCurrencyField

10.15.1 Description

`TCurrencyField` is the field class used by TDataset (250) when it needs to manage currency-valued data.(TField.Datatype (310) equals `ftCurrency`). It simply sets some Tfield (298) 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.

10.15.2 Method overview

Page	Property	Description
236	Create	Create a new instance of a `TCurrencyField`.

10.15.3 Property overview

Page	Property	Access	Description
236	Currency		Is the field a currency field

10.15.4 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 (236)) to be able to work with currency data.

See also: TField (298), TCurrencyField.Currency (236)

10.15.5 TCurrencyField.Currency

Synopsis: Is the field a currency field

Declaration: `Property Currency :`

Visibility: published

Access:

Description: `Currency` is inherited from TFloatField.Currency (335) 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 (335)

10.16 TCustomConnection

10.16.1 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 (238)) and to deactivate the connection (TCustomConnection.Close (237)), plus a property to inspect the state (Connected (239)) of the connected.

10.16.2 Method overview

Page	Property	Description
237	Close	Close the connection
237	Destroy	Remove the `TCustomconnection` instance from memory
238	Open	Makes the connection to the server

10.16.3 Property overview

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

10.16.4 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 (240) 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 (239) event is triggered.

Calling the `Close` method is equivalent to setting the Connected (239) property to `False`.

Errors: If the connection cannot be broken for some reason, an EDatabaseError (220) exception will be raised.

See also: TCustomConnection.BeforeDisconnect (240), TCustomConnection.AfterDisconnect (239), TCustomConnection.Open (238), TCustomConnection.Connected (239)

10.16.5 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 (237)

10.16.6 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 (240) event. If an exception is raised during the execution of that event handler, the connect process is aborted. If LoginPrompt (239) is `True`, the OnLogin (240) event handler is called. Only after this event, the connection is actually established. After the connection was established, the AfterConnect (239) event is triggered.

Calling the `Open` method is equivalent to setting the Connected (239) 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 (240), TCustomConnection.LoginPrompt (239), TCustomConnection.OnLogin (240), TCustomConnection.AfterConnect (239), TCustomConnection.Connected (239)

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

See also: TDataset (250), TCustomConnection.Datasets (238)

10.16.8 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 (238).

See also: TCustomConnection.DatasetCount (238)

10.16.9 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 (238), or to `False` to break it (equivalent to calling TCustomConnection.Close (237).

See also: TCustomConnection.Open (238), TCustomConnection.Close (237)

10.16.10 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 (240)

10.16.11 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 succesfully established in TCustomConnection.Open (238). It can be used to open datasets, or indicate a connection status change.

See also: TCustomConnection.Open (238), TCustomConnection.BeforeConnect (240), TCustomConnection.OnLogin (240)

10.16.12 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 succesfully closed in TCustomConnection.Close (237). It can be used for instance to indicate a connection status change.

See also: TCustomConnection.Close (237), TCustomConnection.BeforeDisconnect (240)

10.16.13 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 (238). 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 (238), TCustomConnection.AfterConnect (239), TCustomConnection.OnLogin (240)

10.16.14 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 (237). It can be used for instance to check for unsaved changes, to save thise 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 (237), TCustomConnection.AfterDisconnect (239)

10.16.15 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 (239) property is `True`, after the TCustomConnection.BeforeConnect (240) event, but before the connection is actually established.

See also: TCustomConnection.BeforeConnect (240), TCustomConnection.LoginPrompt (239), TCustomConnection.Open (238)

10.17 TDatabase

10.17.1 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 (250), 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 (244) 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.

10.17.2 Method overview

Page	Property	Description
242	CloseDataSets	Close all connected datasets
242	CloseTransactions	End all transactions
241	Create	Initialize a new `TDatabase` class instance.
242	Destroy	Remove a `TDatabase` instance from memory.
243	EndTransaction	End an active transaction.
242	StartTransaction	Start a new transaction.

10.17.3 Property overview

Page	Property	Access	Description
244	Connected	rw	Is the datbase connected
244	DatabaseName	rw	Database name or path
243	Directory	rw	Directory for the database
244	IsSQLBased	r	Is the database SQL based.
244	KeepConnection	rw	Should the connection be kept active
245	Params	rw	Connection parameters
243	TransactionCount	r	Number of transaction components connected to this database.
243	Transactions	r	Indexed access to all transaction components connected to this database.

10.17.4 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 (293), TTransaction (208), TDatabase.Destroy (242)

10.17.5 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 (242)

10.17.6 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 (237), TDatabase.CloseTransactions (242)

10.17.7 TDatabase.CloseTransactions

Synopsis: End all transactions

Declaration: `procedure CloseTransactions`

Visibility: public

Description: `CloseTransaction` calls TTransaction.EndTransaction (208) on all connected transactions. It is called automatically when the connection is closed, after all datasets are closed.

See also: TCustomConnection.Close (237), TDatabase.CloseDatasets (242)

10.17.8 TDatabase.StartTransaction

Synopsis: Start a new transaction.

Declaration: `procedure StartTransaction; Virtual; Abstract`

Visibility: public

Description: `StartTransaction` must be implemented by descendent classes to start a new transaction. This method is provided for Delphi compatibility: new applications shoud use a TDBTransaction (294) component instead and invoke the TDBTransaction.StartTRansaction (294) method.

See also: TDBTransaction (294), TDBTransaction.StartTRansaction (294)

10.17.9 TDatabase.EndTransaction

Synopsis: End an active transaction.

Declaration: `procedure EndTransaction; Virtual; Abstract`

Visibility: public

Description: `EndTransaction` must be implemented by descendent classes to end an active transaction. This method is provided for Delphi compatibility: new applications shoud use a TDBTransaction (294) component instead and invoke the TDBTransaction.EndTransaction (294) method.

See also: TDBTransaction (294), TDBTransaction.EndTransaction (294)

10.17.10 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 (243) array property.

See also: TDatabase.Transactions (243)

10.17.11 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 (243)

10.17.12 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 (241).

See also: TDatabase.Params (245), TDatabase.IsSQLBased (244)

10.17.13 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 (245), TDatabase.Directory (243)

10.17.14 TDatabase.Connected

Synopsis: Is the datbase connected

Declaration: `Property Connected : Boolean`

Visibility: published

Access: Read,Write

Description: `Connected` is simply promoted to published property from TCustomConnection.Connected (239).

See also: TCustomConnection.Connected (239)

10.17.15 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 (243) 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 (243), TDatabase.Params (245)

10.17.16 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 (245)

10.17.17 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 (243), TDatabase.DatabaseName (244)

10.18 TDataLink

10.18.1 Description

`TDataLink` is used by GUI controls or datasets in a master-detail relationship to handle data events coming from a TDatasource (287) instance. It is a class that exists for component programmers, application coders should never need to use `TDataLink` or one of it's descendents.

DB-Aware Component coders must use a `TDatalink` instance to handle all communication with a TDataset (250) 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 .

10.18.2 Method overview

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

10.18.3 Property overview

Page	Property	Access	Description
247	Active	r	Is the link active
248	ActiveRecord	rw	Currently active record
248	BOF	r	Is the dataset at the first record
248	BufferCount	rw	Set to the number of record buffers this datalink needs.
249	DataSet	r	Dataset this datalink is connected to
249	DataSource	rw	Datasource this datalink is connected to
249	DataSourceFixed	rw	Can the datasource be changed
249	Editing	r	Is the dataset in edit mode
250	Eof	r	
250	ReadOnly	rw	Is the link readonly
250	RecordCount	r	Number of records in the buffer of the dataset

10.18.4 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 (246)

10.18.5 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 (246)

10.18.6 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 (288) method of the DataSource (249) to which the datalink instance is coupled. If the TDatasource.AutoEdit (289) property is `False` then this operation will not succeed, inless 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 (287), TDatalink.Datasource (249), TDatasource.Edit (288), TDatasource.AutoEdit (289)

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

See also: TDatalink.RecordChange (245)

10.18.8 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 (247)

10.18.9 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 (247)

10.18.10 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 (278) whenever the DataSource (249) property is set.

See also: TDatalink.Datasource (249), TDatalink.ReadOnly (250), TDataset.Active (278)

10.18.11 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 (248), TDatalink.EOF (250)

10.18.12 TDataLink.BOF

Synopsis: Is the dataset at the first record

Declaration: `Property BOF : Boolean`

Visibility: public

Access: Read

Description: `BOF` returns TDataset.BOF (271) if the dataset is available, `True` otherwise.

See also: TDatalink.EOF (250), TDataset.BOF (271)

10.18.13 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 (255), TDatalink.ActiveRecord (248)

10.18.14 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 (249), TDataset (250)

10.18.15 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 (287) instance to get access to the dataset it is connected to. A datalink never points directly to a TDataset (250) instance, always to a datasource. When the datasource is enabled or disabled, all `TDatalink` instances connected to it are enabld or disabled at once.

See also: TDataset (250), TDatasource (287)

10.18.16 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 (249) 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 (249)

10.18.17 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 (246) 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 succesfully set in the editing state.

See also: TDatalink.Edit (246), TDataset.Edit (260)

10.18.18 TDataLink.Eof

Synopsis:

Declaration: `Property Eof : Boolean`

Visibility: public

Access: Read

Description: `EOF` returns TDataset.EOF (273) if the dataset is available, `True` otherwise.

See also: TDatalink.BOF (248), TDataset.EOF (273)

10.18.19 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 (246) 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 (247) property can be used to disable delivey of events to the datalink.

See also: TDatalink.Active (247), TDatalink.edit (246)

10.18.20 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 (248) property: `RecordCount` i s always less than `Buffercount`.

See also: TDatalink.BufferCount (248)

10.19 TDataSet

10.19.1 Description

`TDataset` is the main class of the db unit. This abstract class provides all basic funtionality to access data stored in tabular format: The data consists of records, and the data in each record is organised in several fields.

`TDataset` has a buffer to cache a few records in memory, this buffer is used by `TDatasource` to create the ability to use data-aware components.

`TDataset` is an abstract class, which provides the basic functionality to access, navigate through the data and - in case read-write access is available, edit existing or add new records.

`TDataset` is an abstract class: it does not have the knowledge to store or load the records from whatever medium the records are stored on. Descendants add the functionality to load and save the data. Therefor `TDataset` is never used directly, one always instantiates a descendent class.

Initially, no data is available: the dataset is inactive. The Open (268) 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 (278) property). Likewise, the Close (258) 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 (267), Prev (250), First (263) and Last (266) methods, it is possible to navigate through the records. The EOF (273) property will be `True` if the last row has been reached. Likewise, the BOF (271) property will return `True` if the first record in the dataset has been rechaed when navigating backwards. If both proprties are empty, then there is no data available. For dataset descendents that support counting the number of records, the RecordCount (275) will be zero.

The Append (256) and Insert (265) methods can be used to insert new records to the set of records. The TDataset.Delete (259) statement is used to delete the current record, and the Edit (260) 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 (268) method, or can be undone using the Cancel (257) method.

The data in the various fields properties is available through the Fields (277) 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.

10.19.2 Method overview

Page	Property	Description
255	ActiveBuffer	Currently active memory buffer
256	Append	Append a new record to the data
256	AppendRecord	Append a new record to the dataset and fill with data
257	BookmarkValid	Test whether `ABookMark` is a valid bookmark.
257	Cancel	Cancel the current editing operation
257	CheckBrowseMode	Check whether the dataset is in browse mode.
257	ClearFields	Clear the values of all fields
258	Close	Close the dataset
258	CompareBookmarks	Compare two bookmarks
258	ControlsDisabled	Check whether the controls are disabled
255	Create	Create a new `TDataset` instance
259	CreateBlobStream	Create blob stream
259	CursorPosChanged	Indicate a change in cursor position
259	DataConvert	Convert data from/to native format
259	Delete	Delete the current record.
255	Destroy	Free a `TDataset` instance
260	DisableControls	Disable event propagation of controls
260	Edit	Set the dataset in editing mode.
261	EnableControls	Enable event propagation of controls
261	FieldByName	Search a field by name
261	FindField	Find a field by name
262	FindFirst	Find the first active record (deprecated)
262	FindLast	Find the last active record (deprecated)
262	FindNext	Find the next active record (deprecated)
262	FindPrior	Find the previous active record (deprecated)
263	First	Position the dataset on the first record.
263	FreeBookmark	Free a bookmark obtained with `GetBookmark` (deprecated)
263	GetBookmark	Get a bookmark pointer (deprecated)
264	GetCurrentRecord	Copy the data for the current record in a memory buffer
255	GetFieldData	Get the data for a field
264	GetFieldList	Return field instances in a list
264	GetFieldNames	Return a list of all available field names
264	GotoBookmark	Jump to bookmark
265	Insert	Insert a new record at the current position.
265	InsertRecord	Insert a new record with given values.
265	IsEmpty	Check if the dataset contains no data
265	IsLinkedTo	Check whether a datasource is linked to the dataset
266	IsSequenced	Is the data sequenced
266	Last	Navigate forward to the last record
266	Locate	Locate a record based on some key values
267	Lookup	Search for a record and return matching values.
267	MoveBy	Move the cursor position
267	Next	Go to the next record in the dataset.
268	Open	Activate the dataset: Fetch data into memory.
268	Post	Post pending edits to the database.
269	Prior	Go to the previous record
269	Refresh	Refresh the records in the dataset
269	Resync	Resynchronize the data buffer
256	SetFieldData	Store the data for a field
270	SetFields	Set a number of field values at once
270	Translate	Transliterate a buffer
270	UpdateCursorPos	Update cursor position
270	UpdateRecord	Indicate that the record contents have changed
271	UpdateStatus	Get the update status for the current record

10.19.3 Property overview

Page	Property	Access	Description
278	Active	rw	Is the dataset open or closed.
283	AfterCancel	rw	Event triggered after a Cancel operation.
280	AfterClose	rw	Event triggered after the dataset is closed
283	AfterDelete	rw	
281	AfterEdit	rw	Event triggered after the dataset is put in edit mode.
281	AfterInsert	rw	Event triggered after the dataset is put in insert mode.
280	AfterOpen	rw	Event triggered after the dataset is opened.
282	AfterPost	rw	Event called after changes have been posted to the underlying database
284	AfterRefresh	rw	Event triggered after the data has been refreshed.
284	AfterScroll	rw	Event triggered after the cursor has changed position.
279	AutoCalcFields	rw	How often should the value of calculated fields be calculated
282	BeforeCancel	rw	Event triggered before a Cancel operation.
280	BeforeClose	rw	Event triggered before the dataset is closed.
283	BeforeDelete	rw	Event triggered before a Delete operation.
281	BeforeEdit	rw	Event triggered before the dataset is put in edit mode.
280	BeforeInsert	rw	Event triggered before the dataset is put in insert mode.
279	BeforeOpen	rw	Event triggered before the dataset is opened.
282	BeforePost	rw	Event called before changes are posted to the underlying database
284	BeforeRefresh	rw	Event triggered before the data is refreshed.
283	BeforeScroll	rw	Event triggered before the cursor changes position.
271	BOF	r	Is the cursor at the beginning of the data (on the first record)
271	Bookmark	rw	Get or set the current cursor position
272	CanModify	r	Can the data in the dataset be modified
272	DataSource	r	Datasource this dataset is connected to.
273	DefaultFields	r	Is the dataset using persisten fields or not.
273	EOF	r	Indicates whether the last record has been reached.
274	FieldCount	r	Number of fields
274	FieldDefs	rw	Definitions of available fields in the underlying database
277	Fields	r	Indexed access to the fields of the dataset.
277	FieldValues	rw	Acces to field values based on the field names.
277	Filter	rw	Filter to apply to the data in memory.
278	Filtered	rw	Is the filter active or not.
278	FilterOptions	rw	Options to apply when filtering
274	Found	r	Check success of one of the `Find` methods
275	IsUniDirectional	r	Is the dataset unidirectional (i.e. forward scrolling only)
275	Modified	rw	Was the current record modified ?
285	OnCalcFields	rw	Event triggered when values for calculated fields must be computed.
285	OnDeleteError	rw	Event triggered when a delete operation fails.
286	OnEditError	rw	Event triggered when an edit operation fails.
286	OnFilterRecord	rw	Event triggered to filter records.
286	OnNewRecord	rw	Event triggered when a new record is created.
287	OnPostError	rw	Event triggered when a post operation fails.
276	RecNo	rw	Current record number
275	RecordCount	r	Number of records in the dataset
276	RecordSize	r	Size of the record in memory
276	State	r	Current operational state of the dataset

10.19.4 TDataSet.Create

Synopsis: Create a new `TDataset` instance

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

Visibility: public

Description: `Create` initializes a new TDataset (250) 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 (255)

10.19.5 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 sructures and then calls he inherited destructor.

Errors: An exception may occur during the close operation, in that case, the dataset will not be removed from memory.

See also: TDataset.Close (258), TDataset.Create (255)

10.19.6 TDataSet.ActiveBuffer

Synopsis: Currently active memory buffer

Declaration:
```
function ActiveBuffer : PChar
```

Visibility: public

Description: `ActiveBuffer` points to the currently active memory buffer. It should not be used in application code.

10.19.7 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 (311)

10.19.8 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 (311)

10.19.9 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 (265) 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 (265), TDataset.Edit (260)

10.19.10 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 (270)) and attempts to post the record using TDataset.Post (268). 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 (256), TDataset.SetFields (270), TDataset.Post (268)

10.19.11 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 (263), TDataset.SetBookmark (250), TDataset.FreeBookmark (263), TDataset.BookmarkAvailable (250)

10.19.12 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 (250) and TDataset.OnAfterCancel (250) events. If the dataset wa sin insert mode, then the TDataset.OnBeforeScroll (250) and TDataset.OnAfterScroll (250) 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.

Errors:

See also: TDataset.State (276), TDataset.Append (256), TDataset.Insert (265), TDataset.Edit (260)

10.19.13 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 (220) exception is raised.

See also: TDataset.State (276)

10.19.14 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 (220) exception will be raised.

See also: TDataset.State (276), TField.Clear (303)

10.19.15 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 (250) and TDataset.OnAfterClose (250) 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 (278), TDataset.Open (268)

10.19.16 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 (260) call can be used to disable sending of data events to the controls. The sending can be re-enabled with TDataset.EnableControls (261). 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 (260), TDataset.EnableControls (261)

10.19.17 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 mustbe 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 (257), TDataset.GetBookmark (263), TDataset.SetBookmark (250)

10.19.18 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 applictaion 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 (228) and TBlobField.SaveToStream (229) 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 (228), TBlobField.SaveToStream (229)

10.19.19 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).

10.19.20 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 (255), TDataset.SetFieldData (256)

10.19.21 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 (283), TDataset.BeforeScroll (283), TDataset.AfterDelete (283) and TDataset.AfterScroll (284) 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 (220) exception will be raised.

See also: TDataset.Cancel (257), TDataset.BeforeDelete (283), TDataset.BeforeScroll (283), TDataset.AfterDelete (283), TDataset.AfterScroll (284)

10.19.22 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 (261). 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 (258).

See also: TDataset.EnableControls (261), TDataset.ControlsDisabled (258)

10.19.23 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 (281) and TDataset.AfterEdit (281) 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 (256) instead.

Errors: If the dataset is read-only or not opened, then an EDatabaseError (220) exception will be raised.

See also: TDataset.State (276), TDataset.EOF (273), TDataset.BOF (271), TDataset.Append (256), TDataset.BeforeEdit (281), TDataset.AfterEdit (281)

10.19.24 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 (260) 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 (258).

See also: TDataset.DisableControls (260), TDataset.ControlsDisabled (258)

10.19.25 TDataSet.FieldByName

Synopsis: Search a field by name

Declaration: `function FieldByName(const FieldName: String) : TField`

Visibility: public

Description: `FieldByName` is a shortcut for Fields.FieldByName (331): 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 (220) exception will be raised.

See also: TFields.FieldByname (331), TDataset.FindField (261)

10.19.26 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 (331): it searches for the field with fieldname equalling `FieldName`. The case is performed case-insensitive. The matching field instance is returned, and if no match is found, `Nil` is returned.

See also: TDataset.FieldByname (261), TFields.FindField (331)

10.19.27 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 (263) instead.

See also: TDataset.First (263), TDataset.FindLast (262), TDataset.FindNext (262), TDataset.FindPrior (262)

10.19.28 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 (266) instead.

See also: TDataset.Last (266), TDataset.FindFirst (262), TDataset.FindNext (262), TDataset.FindPrior (262)

10.19.29 TDataSet.FindNext

Synopsis: Find the next active record (deprecated)

Declaration: `function FindNext : Boolean`

Visibility: public

Description: `FindLast` positions the cursor on the next record (taking into account filtering), and returns `True` if the cursor position was changed. This method must be implemented by descendents of `TDataset`: The implementation in `TDataset` always returns `False`, indicating that the position was not changed.

This method is deprecated, use TDataset.Next (267) instead.

See also: TDataset.Next (267), TDataset.FindFirst (262), TDataset.FindLast (262), TDataset.FindPrior (262)

10.19.30 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 (269) instead.

See also: TDataset.Prior (269), TDataset.FindFirst (262), TDataset.FindLast (262), TDataset.FindPrior (262)

10.19.31 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 (283) and TDataset.AfterScroll (284) events. After the action is completed, the TDataset.BOF (271) property will be `True`.

Errors: If the dataset is unidirectional or is closed, an EDatabaseError (220) exception will be raised.

See also: TDataset.Prior (269), TDataset.Last (266), TDataset.Next (267), TDataset.BOF (271), TDataset.BeforeScroll (283), TDataset.AfterScroll (284)

10.19.32 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 (263). It should not be used on bookmarks obtained with the TDataset.Bookmark (271) 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 (263), TDataset.Bookmark (271)

10.19.33 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 (250) 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 (263) call. The bookmark will be `Nil` if the dataset is empty or not active.

This call is deprecated. Use the TDataset.Bookmark (271) property instead to get a bookmark.

See also: TDataset.SetBookmark (250), TDataset.FreeBookmark (263), TDataset.Bookmark (271)

10.19.34 TDataSet.GetCurrentRecord

Synopsis: Copy the data for the current record in a memory buffer

Declaration: `function GetCurrentRecord(Buffer: PChar) : 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 succesfully to the buffer, the function returns `True`. The `TDataset` implementation is empty, and returns `False`.

See also: TDataset.ActiveBuffer (255)

10.19.35 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 (220) exception will be raised.

See also: TDataset.GetFieldNames (264), TDataset.FieldByName (261), TDataset.FindField (261)

10.19.36 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 (250), TDataset.FieldByName (261), TDataset.FindField (261)

10.19.37 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 (263) function.

This function is deprecated, use the TDataset.Bookmark (271) property instead.

Errors: if `ABookmark` does not contain a valid bookmark, then an exception may be raised.

See also: TDataset.Bookmark (271), TDataset.GetBookmark (263), TDataset.FreeBookmark (263)

10.19.38 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 (280), BeforeScroll (283), OnNewRecord (286), AfterInsert (281) and AfterScroll (284) 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 (268) must be called.

Errors: If the dataset is read-only, calling `Insert` will result in an EDatabaseError (220).

See also: TDataset.BeforeInsert (280), TDataset.BeforeScroll (283), TDataset.OnNewrecord (286), TDataset.AfterInsert (281), TDataset.AfterScroll (284), TDataset.Post (268), TDataset.Append (256)

10.19.39 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.

Errors:

See also: TDataset.Insert (265), TDataset.SetFieldValues (250)

10.19.40 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 (273) and TDataset.BOF (271) are both `True`, and the dataset is not in insert mode.

See also: TDataset.EOF (273), TDataset.BOF (271), TDataset.State (276)

10.19.41 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" (289) points to the current dataset instance, or indirectly.

See also: TDatasource.Dataset (289)

10.19.42 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 (276) 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 (276)

10.19.43 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. After a call to `Last`, the TDataset.EOF (273) property will be `True`. Calling this method will trigger the TDataset.BeforeScroll (283) and TDataset.AfterScroll (284) events.

See also: TDataset.First (263), TDataset.Next (267), TDataset.EOF (273), TDataset.BeforeScroll (283), TDataset.AfterScroll (284)

10.19.44 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 (216).

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 (250), TDataset.Lookup (267), TLocateOption (216)

10.19.45 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 (266) 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 (220) exception will be raised.

See also: TDataset.Locate (266)

10.19.46 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 (283) and TDataset.AfterScroll (284) 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 (220) exception on unidirectional datasets.

See also: TDataset.RecNo (276), TDataset.BeforeScroll (283), TDataset.AfterScroll (284)

10.19.47 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 (283) and TDataset.AfterScroll (284) events. If the dataset is located on the last known record (EOF (273) 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 (220) exception.

See also: TDataset.MoveBy (267), TDataset.Prior (269), TDataset.Last (266), TDataset.BeforeScroll (283), TDataset.AfterScroll (284), TDataset.EOF (273)

10.19.48 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 (271)is set to `True`
4. Internal buffers are allocated and filled with data
5. If the dataset is empty, EOF (273) is set to true
6. State (276) is set to `dsBrowse`
7. The AfterOpen (280) 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 (278), TDataset.State (276), TDataset.BOF (271), TDataset.EOF (273), TDataset.BeforeOpen (279), TDataset.AfterOpen (280)

10.19.49 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 (265), Append (256) or TDataset.Edit (260) 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 (282), AfterPost (282) events. After the call to `Past`, the State (276) 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 (209)) will result in an EdatabaseError (220) exception. If an exception occurs during the save operation, the OnPostError (287) event is triggered to handle the error.

See also: TDataset.Insert (265), Tdataset.Append (256), TDataset.Edit (260), Tdataset.OnPostError (287), TDataset.BeforePost (282), TDataset.AfterPost (282), TDataset.State (276)

10.19.50 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 (283) and TDataset.AfterScroll (284) events. If the dataset is located on the first record, (BOF (271) 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 (220) exception.

See also: TDataset.MoveBy (267), TDataset.Next (267), TDataset.First (263), TDataset.BeforeScroll (283), TDataset.AfterScroll (284), TDataset.BOF (271)

10.19.51 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 (284) and AfterRefresh (284) events.

Errors: Refreshing may fail if the underlying dataset descendent does not support it.

See also: TDataset.Close (258), TDataset.Open (268), TDataset.BeforeRefresh (284), TDataset.AfterRefresh (284)

10.19.52 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 (269) should be used. The `Resync` parameter indicates how the buffers should be refreshed.

See also: TDataset.Refresh (269)

10.19.53 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 (277) property, and works it's way down the array.

Errors: If the dataset is not in edit mode, then an EDatabaseError (220) exception will be raised. If there are more values than fields, an `EListError` exception will be raised.

See also: TDataset.Fields (277)

10.19.54 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 (374) property is set to `True`. The `toOEM` parameter is set to `True` if the transliteration must happen from the used codepage to the codepage used for storage, and if it is set to `False` then the transliteration must happen from the native codepage to the storage codepage. This call must be overridden by descendents of `TDataset` to provide the necessary transliteration: `TDataset` just copies the contents of the `Src` buffer to the `Dest` buffer. The result must be the number of bytes copied to the destination buffer.

Errors: No checks are performed on the bufffers.

See also: TStringField.Transliterate (374)

10.19.55 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 (269)

10.19.56 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: TDataset.OnUpdateRecord (250)

10.19.57 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.

10.19.58 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 (263). As soon as TDataset.Next (267) is called, `BOF` will no longer be true.

See also: TDataset.EOF (273), TDataset.Next (267), TDataset.First (263)

10.19.59 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;
```

At the end of this code, the dataset will be positioned on the same record as when the code was started. The TDataset.DisableControls (260) and TDataset.EnableControls (261) calls prevent the controls from receiving update notifications as the dataset scrolls through the records, tus recuding 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 (260), TDataset.EnableControls (261)

10.19.60 TDataSet.CanModify

Synopsis: Can the data in the dataset be modified

Declaration: `Property CanModify : Boolean`

Visibility: public

Access: Read

Description: `CanModifiy` indicates whether the dataset allows editing. `Unidirectional` datasets do not allow editing. Descendent datasets can impose additioanl 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 (265), TDataset.Append (256), TDataset.Delete (259), Tdataset.Edit (260)

10.19.61 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 (287)

10.19.62 TDataSet.DefaultFields

Synopsis: Is the dataset using persisten 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 desin time with the fields editor. If `DefaultFields` is `True`, then for each item in the TDataset.FieldDefs (274) 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 (274), TDataset.Fields (277), TField (298)

10.19.63 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 TDataset.Last (266) method is called.
2. The record is on the last record, and the TDataset.Next (267) method is called.
3. The dataset is empty when opened

In all other cases, `EOF` is `False`. Note that when the cursor is on the last-but-one record, and `Next` is called, `EOF` will not yet be `True`. It is only when the cursor is on the last record and `Next` is called, that `EOF` will 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 (271), TDataset.Next (267), TDataset.Last (266), TDataset.IsEmpty (265)

10.19.64 TDataSet.FieldCount

Synopsis: Number of fields

Declaration: `Property FieldCount : LongInt`

Visibility: public

Access: Read

Description: `FieldCount` is the same as Fields.Count (333), i.e. the number of fields. For a dataset with persistent fields (when DefaultFields (273) 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 (329)

10.19.65 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 (273) is `True`), then for each item in this list, a field i nstance 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 (268), TDataset.DefaultFields (273), TDataset.Fields (277)

10.19.66 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 (262),TDataset.FindLast (262),TDataset.FindNext (262) or TDataset.FindPrior (262) operations was succesful.

See also: TDataset.FindFirst (262), TDataset.FindLast (262), TDataset.FindNext (262), TDataset.FindPrior (262)

10.19.67 TDataSet.Modified

Synopsis: Was the current record modified ?

Declaration: `Property Modified : Boolean`

Visibility: public

Access: Read,Write

Description: `Modified` is `True` if the current record was modified after a call to Tdataset.Edit (260) or Tdataset.Insert (265). It becomes `True` if a value was written to one of the fields of the dataset.

See also: Tdataset.Edit (260), TDataset.Insert (265), TDataset.Append (256), TDataset.Cancel (257), TDataset.Post (268)

10.19.68 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 (269) or TDataset.Next (267)), random positioning of the cursor, editing or filtering will result in an EDatabaseError (220). 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 (269), TDataset.Next (267)

10.19.69 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 (276)

10.19.70 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 (275)

10.19.71 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 (310), TDataset.RecordCount (275), TDataset.RecNo (276)

10.19.72 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 (278), TDataset.Edit (260), TDataset.Insert (265), TDataset.Append (256), TDataset.Post (268), TDataset.Cancel (257)

10.19.73 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 (329) 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 (326), TField (298)

10.19.74 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: FieldByname (208)

10.19.75 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 (278), TDataset.FilterOptions (278)

10.19.76 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 (277) 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 (277), TDataset.FilterOptions (278)

10.19.77 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 (277), 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 (277), TDataset.FilterOptions (278)

10.19.78 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 (276) is then `dsInactive`). Setting the `Active` property to `True` is equivalent to calling TDataset.Open (268), setting it to `False` is equivalent to calling TDataset.Close (258)

See also: TDataset.State (276), TDataset.Open (268), TDataset.Close (258)

10.19.79 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 (285) event.

See also: TField.FieldKind (316), TDataset.OnCalcFields (285)

10.19.80 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 (280), TDataset.State (276)

10.19.81 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 (284) event will be triggered immediatly 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 (280), TDataset.State (276), TDataset.AfterScroll (284)

10.19.82 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 (280), TDataset.State (276)

10.19.83 TDataSet.AfterClose

Synopsis: Event triggered after the dataset is closed

Declaration: `Property AfterClose : TDataSetNotifyEvent`

Visibility: public

Access: Read,Write

Description: `AfterOpen` is triggered after the dataset is opened. The dataset has discarded its data and has cleaned up it's internal memory structures. It is in `dsInactive` state when this event is triggered.

See also: TDataset.BeforeClose (280), TDataset.State (276)

10.19.84 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 (256) or TDataset.Insert (265) 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 (281), TDataset.Append (256), TDataset.Insert (265)

10.19.85 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 (284) 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 (280), TDataset.AfterScroll (284), TDataset.Append (256), TDataset.Insert (265)

10.19.86 TDataSet.BeforeEdit

Synopsis: Event triggered before the dataset is put in edit mode.

Declaration: `Property BeforeEdit : TDataSetNotifyEvent`

Visibility: public

Access: Read,Write

Description: `BeforeEdit` is the triggered at the start of the TDataset.Edit (260) 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 (281), TDataset.Edit (260), TDataset.State (276)

10.19.87 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 (281), TDataset.Edit (260), TDataset.State (276)

10.19.88 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 (268) method, when the datset 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 (268), TDataset.AfterPost (282), TDataset.State (276)

10.19.89 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 (268) operation was succesfully completed, and the dataset is again in `dsBrowse` state. If an error occured during the post operation, then the `AfterPost` event is not called, but the TDataset.OnPostError (287) event is triggered instead.

See also: TDataset.BeforePost (282), TDataset.Post (268), TDataset.State (276), TDataset.OnPostError (287)

10.19.90 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 (257) 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 (283), TDataset.Cancel (257), TDataset.State (276)

10.19.91 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 (257) operation was succesfully completed, and the dataset is again in `dsBrowse` state.

See also: TDataset.BeforeCancel (282), TDataset.Cancel (257), TDataset.State (276)

10.19.92 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 (259) 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 (283) event. If the dataset was in insert mode when the `Delete` method was called, then the event will not be called, as TDataset.Cancel (257) is called instead.

See also: TDataset.AfterDelete (283), TDataset.Delete (259), TDataset.BeforeScroll (283), TDataset.Cancel (257), TDataset.State (276)

10.19.93 TDataSet.AfterDelete

Synopsis:

Declaration: `Property AfterDelete : TDataSetNotifyEvent`

Visibility: public

Access: Read,Write

Description: `AfterDelete` is triggered after the succesfull completion of the TDataset.Delete (259) operation, when the dataset is again in `dsBrowse` state. The event is followed by a TDataset.AfterScroll (284) event.

See also: TDataset.BeforeDelete (283), TDataset.Delete (259), TDataset.AfterScroll (284), TDataset.State (276)

10.19.94 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 (267), TDataset.Prior (269), TDataset.First (263), TDataset.Last (266), but also with two of the editing operations:TDataset.Insert (265) and TDataset.Delete (259). Raising an exception in this event handler aborts the operation in progress.

See also: TDataset.AfterScroll (284), TDataset.Next (267), TDataset.Prior (269), TDataset.First (263), TDataset.Last (266), TDataset.Insert (265), TDataset.Delete (259)

10.19.95 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 (267), TDataset.Prior (269), TDataset.First (263), TDataset.Last (266), but also with two of the editing operations:TDataset.Insert (265) and TDataset.Delete (259) 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 (284), TDataset.Next (267), TDataset.Prior (269), TDataset.First (263), TDataset.Last (266), TDataset.Insert (265), TDataset.Delete (259), TDataset.Open (268)

10.19.96 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 (269) method, after the dataset has been put in browse mode. If the dataset cannot be put in browse mode, the `BeforeRefresh` method wil 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 (269), TDataset.AfterRefresh (284), TDataset.State (276)

10.19.97 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 (269) 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 (269), TDataset.State (276), TDataset.BeforeRefresh (284)

10.19.98 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 (279) 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 (279), TField.Kind (298)

10.19.99 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 (259) 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 (212) event handler type.

See also: TDatasetErrorEvent (212), TDataset.Delete (259), TDataset.OnEditError (286), TDataset.OnPostError (287)

10.19.100 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 (260) 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 (212) event handler type.

See also: TDatasetErrorEvent (212), TDataset.Edit (260), TDataset.OnDeleteError (285), TDataset.OnPostError (287)

10.19.101 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 (278) 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 (277), TDataset.Filtered (278), TDataset.state (276)

10.19.102 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 (256) or TDataset.Insert (265) 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 (276) is again `dsBrowse`).

See also: TDataset.Append (256), TDataset.Insert (265), TDataset.State (276)

10.19.103 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 (268) 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 (212) event handler type.

See also: TDatasetErrorEvent (212), TDataset.Post (268), TDataset.OnDeleteError (285), TDataset.OnEditError (286)

10.20 TDataSource

10.20.1 Description

`TDatasource` is a mediating component: it handles communication between any DB-Aware component (often edit controls on a form) and a TDataset (250) instance. Any database aware component should never communicate with a dataset directly. Instead, it should communicate with a TDatasource (287) 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.

10.20.2 Method overview

Page	Property	Description
288	Create	Create a new instance of `TDatasource`
288	Destroy	Remove a `TDatasource` instance from memory
288	Edit	Put the dataset in edit mode, if needed
289	IsLinkedTo	Check if a dataset is linked to a certain dataset

10.20.3 Property overview

Page	Property	Access	Description
289	AutoEdit	rw	Should the dataset be put in edit mode automatically
289	DataSet	rw	Dataset this datasource is connected to
290	Enabled	rw	Enable or disable sending of events
290	OnDataChange	rw	Called whenever data changes in the current record
290	OnStateChange	rw	Called whenever the state of the dataset changes
291	OnUpdateData	rw	Called whenever the data in the dataset must be updated
289	State	r	State of the dataset

10.20.4 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 (288)

10.20.5 TDataSource.Destroy

Synopsis: Remove a `TDatasource` instance from memory

Declaration: `destructor Destroy; Override`

Visibility: public

Description: `Destroy` notifies all TDataLink (245) instances connected to it that the dataset is no loner available, and then removes itself from the `TDatalink` instance. It then cleans up all resources and calls the inherited constructor.

See also: TDatasource.Create (288), TDatalink (245)

10.20.6 TDataSource.Edit

Synopsis: Put the dataset in edit mode, if needed

Declaration: `procedure Edit`

Visibility: public

Description: `Edit` will check AutoEdit (289): if it is `True`, then it puts the Dataset (289) 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 (260) directly.

Errors: An EDatabaseError (220) 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 (289), TDataset.Edit (260), TDataset.State (276)

10.20.7 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 (289) property, and returns `True` if it is the same. If not, it continues by checking any detail dataset fields that the dataset posesses (recursively). This function can be used to detect circular links in e.g. master-detail relationships.

See also: TDatasource.Dataset (289)

10.20.8 TDataSource.State

Synopsis: State of the dataset

Declaration: `Property State : TDataSetState`

Visibility: public

Access: Read

Description: `State` contains the State (276) 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 (276) directly, to take into account the effect of the Enabled (290) property.

See also: TDataset.State (276), TDatasource.Enabled (290)

10.20.9 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 (288) 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 (288), TDataset.Edit (260)

10.20.10 TDataSource.DataSet

Synopsis: Dataset this datasource is connected to

Declaration: `Property DataSet : TDataSet`

Visibility: published

Access: Read,Write

Description: `Dataset` must be set by the applictaion programmer to the TDataset (250) 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 (250), TDatasource.Enabled (290)

10.20.11 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 (250), TDatasource.Dataset (289), TDatasource.AutoEdit (289)

10.20.12 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 (276) 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 (291), TDatasource.OnStateChange (290), TDataset.State (276), TDatasource.Enabled (290)

10.20.13 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 (291), TDatasource.OnStateChange (290), TDataset.AfterScroll (284), TField.OnChange (321), TDatasource.Enabled (290)

10.20.14 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 (268) 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 (290), TDatasource.OnStateChange (290), TDataset.Post (268)

10.21 TDateField

10.21.1 Description

`TDateField` is the class used when a dataset must manage data of type date. (TField.DataType (310) equals `ftDate`). It initializes some of the properties of the TField (298) 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.

10.21.2 Method overview

Page	Property	Description
291	Create	Create a new instance of a `TDateField` class.

10.21.3 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 (298) properties to configure the instance for working with date values.

See also: TField (298)

10.22 TDateTimeField

10.22.1 Description

`TDateTimeField` is the class used when a dataset must manage data of type datetime. (TField.DataType (310) equals `ftDateTime`). It also serves as base class for the TDateField (291) or TTimeField (375) classes. It overrides some of the properties and methods of the TField (298) 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.

10.22.2 Method overview

Page	Property	Description
292	Create	Create a new instance of a `TDateTimeField` class.

10.22.3 Property overview

Page	Property	Access	Description
292	DisplayFormat	rw	Formatting string for textual representation of the field
292	Value	rw	Contents of the field as a TDateTime value

10.22.4 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 (298) properties to configure the instance for working with date/time values.

See also: TField (298)

10.22.5 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 (314) by `TDateTimeField` as a `TDateTime` value. It returns the same value as the TField.AsDateTime (306) property.

See also: TField.AsDateTime (306), TField.Value (314)

10.22.6 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 (311) property to format the value with the DateTimeToString (**??**)function.

See also: #rtl.sysutils.datetimetostring (**??**), #rtl.sysutils.formatdatetime (**??**), TField.DisplayText (311)

10.23 TDBDataset

10.23.1 Description

`TDBDataset` is a `TDataset` descendent which introduces the concept of a database: a central component (TDatabase (241)) which represents a connection to a database. This central component is exposed in the TDBDataset.Database (293) 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 (294) 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`.

10.23.2 Method overview

Page	Property	Description
293	destroy	Remove the `TDBDataset` instance from memory.

10.23.3 Property overview

Page	Property	Access	Description
293	DataBase	rw	Database this dataset is connected to
294	Transaction	rw	Transaction in which this dataset is running.

10.23.4 TDBDataset.destroy

Synopsis: Remove the `TDBDataset` instance from memory.

Declaration: `destructor destroy;  Override`

Visibility: public

Description: `Destroy` will disconnect the `TDBDataset` from its Database (293) and Transaction (294). After this it calls the inherited destructor.

See also: TDBDataset.Database (293), TDatabase (241)

10.23.5 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 (241) 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 (220) exception will be raised.

See also: TDatabase (241), TDBDataset.Transaction (294)

10.23.6 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 (294) 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 (294), TDBDataset.Database (293)

10.24 TDBTransaction

10.24.1 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 (241) instance, and to keep track of `TDataset` instances which are connected to the transaction.

10.24.2 Method overview

Page	Property	Description
295	CloseDataSets	Close all connected datasets
294	Create	Transaction property
295	destroy	Remove a `TDBTransaction` instance from memory.

10.24.3 Property overview

Page	Property	Access	Description
295	Active	rw	Is the transaction active or not
295	DataBase	rw	Database this transaction is connected to

10.24.4 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 (295)

10.24.5 TDBTransaction.destroy

Synopsis: Remove a `TDBTransaction` instance from memory.

Declaration: `destructor destroy; Override`

Visibility: public

Description: `Destroy` first disconnects all connected TDBDataset (293) instances and then cleans up the resources allocated in the Create (294) constructor. After that it calls the inherited destructor.

See also: TDBTransaction.Create (294)

10.24.6 TDBTransaction.CloseDataSets

Synopsis: Close all connected datasets

Declaration: `procedure CloseDataSets`

Visibility: public

Description: `CloseDatasets` closes all connected datasets (All TDBDataset (293) instances whose Transaction (294) property points to this `TDBTransaction` instance).

See also: TDBDataset (293), TDBDataset.Transaction (294)

10.24.7 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 (220) exception.

See also: TDBTransaction.Active (295), TDatabase (241)

10.24.8 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 (294). Reversely, setting `Active` to `True` will call `StartTransaction`, setting it to `False` will call TDBTransaction.EndTransaction (294).

See also: TDBTransaction.StartTransaction (294), TDBTransaction.EndTransaction (294)

10.25 TDefCollection

10.25.1 Description

`TDefCollection` is a parent class for the TFieldDefs (326) and TIndexDefs (341) collections: It holds a set of named definitions on behalf of a TDataset (250) component. To this end, it introduces a dataset (297) property, and a mechanism to notify the dataset of any updates in the collection. It is supposed to hold items of class TNamedItem (354), so the TDefCollection.Find (296) method can find items by named.

10.25.2 Method overview

Page	Property	Description
296	create	Instantiate a new `TDefCollection` instance.
296	Find	Find an item by name
297	GetItemNames	Return a list of all names in the collection
297	IndexOf	Find location of item by name

10.25.3 Property overview

Page	Property	Access	Description
297	Dataset	r	Dataset this collection manages definitions for.
297	Updated	rw	Has one of the items been changed

10.25.4 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 (250), TNamedItem (354)

10.25.5 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 (354), TDefCollection.IndexOf (297)

10.25.6 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 (354)

10.25.7 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 (296), TNamedItem.Name (354)

10.25.8 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 cgange during the lifetime of the collection.

10.25.9 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.

10.26 TDetailDataLink

10.26.1 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.

10.26.2 Property overview

Page	Property	Access	Description
298	DetailDataSet	r	Detail dataset in Master-detail relation

10.26.3 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 (349). The master dataset is available through the regular TDataLink.DataSource (249) property.

See also: TDataset (250), TMasterDatalink (349), TDataLink.DataSource (249)

10.27 TField

10.27.1 Description

`TField` is an abstract class that defines access methods for a field in a record, controlled by a TDataset (250) 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.

10.27.2 Method overview

Page	Property	Description
302	Assign	Copy properties from one TField instance to another
302	AssignValue	Assign value of a variant record to the field.
303	Clear	Clear the field contents.
302	Create	Create a new `TField` instance
302	Destroy	Destroy the `TField` instance
303	FocusControl	Set focus to the first control connected to this field.
303	GetData	Get the data from this field
304	IsBlob	Is the field a BLOB field (untyped data of indeterminate size).
304	IsValidChar	Check whether a character is valid input for the field
304	RefreshLookupList	Refresh the lookup list
304	SetData	Save the field data
305	SetFieldType	Set the field data type
305	Validate	Validate the data buffer

10.27.3 Property overview

Page	Property	Access	Description
315	Alignment	rw	Alignment for this field
305	AsBCD	rw	Access the field's contents as a BCD (Binary coded Decimal)
306	AsBoolean	rw	Access the field's contents as a Boolean value.
306	AsCurrency	rw	Access the field's contents as a Currency value.
306	AsDateTime	rw	Access the field's contents as a TDateTime value.
307	AsFloat	rw	Access the field's contents as a floating-point (Double) value.
308	AsInteger	rw	Access the field's contents as a 32-bit signed integer (longint) value.
307	AsLargeInt	rw	Access the field's contents as a 64-bit signed integer (longint) value.
307	AsLongint	rw	Access the field's contents as a 32-bit signed integer (longint) value.
308	AsString	rw	Access the field's contents as an AnsiString value.
309	AsVariant	rw	Access the field's contents as a Variant value.
308	AsWideString	rw	Access the field's contents as a WideString value.
309	AttributeSet	rw	
309	Calculated	rw	Is the field a calculated field ?
309	CanModify	r	Can the field's contents be modified.
315	ConstraintErrorMessage	rw	Message to display if the `CustomConstraint` constraint is violated.
310	CurValue	r	Current value of the field
315	CustomConstraint	rw	Custom constraint for the field's value
310	DataSet	rw	Dataset this field belongs to
310	DataSize	r	Size of the field's data
310	DataType	r	The data type of the field.
316	DefaultExpression	rw	Default value for the field
316	DisplayLabel	rws	Name of the field for display purposes
311	DisplayName	r	User-readable fieldname
311	DisplayText	r	Formatted field value
316	DisplayWidth	rw	Width of the field in characters
316	FieldKind	rw	The kind of field.
317	FieldName	rw	Name of the field
311	FieldNo	r	Number of the field in the record
317	HasConstraints	r	Does the field have any constraints defined
317	ImportedConstraint	rw	Constraint for the field value on the level of the underlying database
317	Index	rw	Index of the field in the list of fields
312	IsIndexField	r	Is the field an indexed field ?
312	IsNull	r	Is the field empty
318	KeyFields	rw	Key fields to use when looking up a field value.
312	Lookup	rw	Is the field a lookup field
318	LookupCache	rw	Should lookup values be cached
318	LookupDataSet	rw	Dataset with lookup values
319	LookupKeyFields	rw	Names of fields on which to perform a locate
314	LookupList	r	List of lookup values
319	LookupResultField	rw	Name of field to use as lookup value
312	NewValue	rw	The new value of the field
313	Offset	r	Offset of the field's value in the dataset buffer
314	OldValue	r	Old value of the field
321	OnChange	rw	Event triggerd when the field's value has changed
321	OnGetText	rw	Event to format the field's content
321	OnSetText	rw	Event to set the field's content based on a user-formatted string
322	OnValidate	rw	Event to validate the value of a field before it is written to the data buffer
319	Origin	rw	Original fieldname of the field.
319	ProviderFlags	rw	Flags for provider or update support

10.27.4 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 (268) method will create the necessary `TField` instances, if none had been creaed in the designer.

See also: TDataset.Open (268)

10.27.5 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 (258)

10.27.6 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 (314)

10.27.7 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 (220) exception.

See also: TField.Assign (302), TField.Value (314)

10.27.8 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 (312) 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 (220) exception.

See also: TField.IsNull (312), TField.Value (314)

10.27.9 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 (261), TDataset.DisableControls (260)

10.27.10 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 (312), TField.SetData (304), TField.Value (314)

10.27.11 TField.IsBlob

Synopsis: Is the field a BLOB field (untyped data of indeterminate size).

Declaration:
```
function IsBlob : Boolean;  Virtual
```

Visibility: public

Description: `IsBlob` returns `True` if the field is one of the blob field types. The `TField` implementation returns false. Only one of the blob-type field classes override this function and let it return `True`.

Errors: None.

See also: TBlobField.IsBlob (228)

10.27.12 TField.IsValidChar

Synopsis: Check whether a character is valid input for the field

Declaration:
```
function IsValidChar(InputChar: Char) : Boolean;  Virtual
```

Visibility: public

Description: `IsValidChar` checks whether `InputChar` is a valid characters for the current field. It does this by checking whether `InputChar` is in the set of characters sepcified by the TField.ValidChars (313) 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 (313)

10.27.13 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 (318) 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: TField.LookupDataset (318), TField.LookupKeyFields (319), TField.LookupResultField (319)

10.27.14 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 (303), TField.Value (314)

10.27.15 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 (310)

10.27.16 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 (322) event handler, if one is set, to allow the application programmer to program additional checks.

See also: TField.SetData (304), TField.OnValidate (322)

10.27.17 TField.AsBCD

Synopsis: Access the field's contents as a BCD (Binary coded Decimal)

Declaration: `Property AsBCD : TBCD`

Visibility: public

Access: Read,Write

Description: `AsBCD` can be used to read or write the contents of the field as a BCD value (Binary Coded Decimal). If the native type of the field is not BCD, then an attempt will be made to convert the field value from the native format to a BCD value when reading the field's content. Likewise, when writing the property, the value will be converted to the native type of the field (if the value allows it). Therefor, when reading or writing a field value for a field whose native data type is not a BCD value, an exception may be raised.

See also: TField.AsCurrency (306), TField.Value (314)

10.27.18 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 (314), TField.AsInteger (308)

10.27.19 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 (314), TField.AsFloat (307)

10.27.20 TField.AsDateTime

Synopsis: Access the field's contents as a TDateTime value.

Declaration: `Property AsDateTime : TDateTime`

Visibility: public

Access: Read,Write

Description: `AsDateTime` can be used to read or write the contents of the field as a TDateTime value (for both date and time values). If the native type of the field is not a date or time value, then an attempt will be made to convert the field value from the native format to a TDateTime value when reading the field's content. Likewise, when writing the property, the value will be converted to the native type of the field (if the value allows it). Therefor, when reading or writing a field value for a field whose native data type is not a TDateTime-compatible value (dates or string values), an exception may be raised.

See also: TField.Value (314), TField.AsString (308)

10.27.21 TField.AsFloat

Synopsis: Access the field's contents as a floating-point (Double) value.

Declaration: `Property AsFloat : Double`

Visibility: public

Access: Read,Write

Description: `AsFloat` can be used to read or write the contents of the field as a floating-point value (of type double, i.e. with double precision). If the native type of the field is not a floating-point value, then an attempt will be made to convert the field value from the native format to a floating-point value when reading the field's content. Likewise, when writing the property, the value will be converted to the native type of the field (if the value allows it). Therefor, when reading or writing a field value for a field whose native data type is not a floating-point-compatible value (string values for instance), an exception may be raised.

See also: TField.Value (314), TField.AsString (308), TField.AsCurrency (306)

10.27.22 TField.AsLongint

Synopsis: Access the field's contents as a 32-bit signed integer (longint) value.

Declaration: `Property AsLongint : LongInt`

Visibility: public

Access: Read,Write

Description: `AsLongint` can be used to read or write the contents of the field as a 32-bit signed integer value (of type longint). If the native type of the field is not a longint value, then an attempt will be made to convert the field value from the native format to a longint value when reading the field's content. Likewise, when writing the property, the value will be converted to the native type of the field (if the value allows it). Therefor, when reading or writing a field value for a field whose native data type is not a 32-bit signed integer-compatible value (string values for instance), an exception may be raised.

This is an alias for the TField.AsInteger (308).

See also: TField.Value (314), TField.AsString (308), TField.AsInteger (308)

10.27.23 TField.AsLargeInt

Synopsis: Access the field's contents as a 64-bit signed integer (longint) value.

Declaration: `Property AsLargeInt : LargeInt`

Visibility: public

Access: Read,Write

Description: `AsLargeInt` can be used to read or write the contents of the field as a 64-bit signed integer value (of type Int64). If the native type of the field is not an Int64 value, then an attempt will be made to convert the field value from the native format to an Int64 value when reading the field's content. Likewise, when writing the property, the value will be converted to the native type of the field (if the value allows it). Therefor, when reading or writing a field value for a field whose native data type is not a 64-bit signed integer-compatible value (string values for instance), an exception may be raised.

See also: TField.Value (314), TField.AsString (308), TField.AsInteger (308)

10.27.24 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). Therefor, when reading or writing a field value for a field whose native data type is not a 32-bit signed integer-compatible value (string values for instance), an exception may be raised.

See also: TField.Value (314), TField.AsString (308), TField.AsLongint (307), TField.AsInt64 (298)

10.27.25 TField.AsString

Synopsis: Access the field's contents as an AnsiString value.

Declaration: `Property AsString : String`

Visibility: public

Access: Read,Write

Description: `AsString` can be used to read or write the contents of the field as an AnsiString value. If the native type of the field is not an ansistring value, then an attempt will be made to convert the field value from the native format to a ansistring value when reading the field's content. Likewise, when writing the property, the value will be converted to the native type of the field (if the value allows it). Therefor, when reading or writing a field value for a field whose native data type is not an ansistring-compatible value, an exception may be raised.

See also: TField.Value (314), TField.AsWideString (308)

10.27.26 TField.AsWideString

Synopsis: Access the field's contents as a WideString value.

Declaration: `Property AsWideString : WideString`

Visibility: public

Access: Read,Write

Description: `AsWideString` can be used to read or write the contents of the field as a WideString value. If the native type of the field is not a widestring value, then an attempt will be made to convert the field value from the native format to a widestring value when reading the field's content. Likewise, when writing the property, the value will be converted to the native type of the field (if the value allows it). Therefor, when reading or writing a field value for a field whose native data type is not a widestring-compatible value, an exception may be raised.

See also: TField.Value (314), TField.Astring (298)

10.27.27 TField.AsVariant

Synopsis: Access the field's contents as a Variant value.

Declaration: `Property AsVariant : variant`

Visibility: public

Access: Read,Write

Description: `AsVariant` can be used to read or write the contents of the field as a Variant value. If the native type of the field is not a Variant value, then an attempt will be made to convert the field value from the native format to a variant value when reading the field's content. Likewise, when writing the property, the value will be converted to the native type of the field (if the value allows it). Therefor, when reading or writing a field value for a field whose native data type is not a variant-compatible value, an exception may be raised.

See also: TField.Value (314), TField.Astring (298)

10.27.28 TField.AttributeSet

Synopsis:

Declaration: `Property AttributeSet : String`

Visibility: public

Access: Read,Write

Description:

10.27.29 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 (316) is `fkCalculated`. Setting the property wil 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 (316)

10.27.30 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 (320), TDataset.CanModify (272)

10.27.31 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 (314)

10.27.32 TField.DataSet

Synopsis: Dataset this field belongs to

Declaration: `Property DataSet : TDataSet`

Visibility: public

Access: Read,Write

Description: `Dataset` contains the dataset this field belongs to. Writing this property will add the field to the list of fields of a dataset, after removing if from the list of fields of the dataset the field was previously assigned to. It should under normal circumstnces never be necessary to set this property, the `TDataset` code will take care of this.

See also: TDataset (250), TDataset.Fields (277)

10.27.33 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 (313) property which declares a logical size for datatypes that have a variable size (such as string fields). For BLOB fields, use the TBlobField.BlobSize (229) property to get the size of the field's contents for the current record..

See also: TField.Size (313), TBlobField.BlobSize (229)

10.27.34 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 (316)

10.27.35 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 (317) property, unless assigned another value.

The use of this property is deprecated. Use DisplayLabel (316) instead.

See also: Tfield.FieldName (317)

10.27.36 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 (308) property, which does not take into account any formatting.

See also: TField.AsString (308)

10.27.37 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 (317)

10.27.38 TField.IsIndexField

Synopsis: Is the field an indexed field ?

Declaration: `Property IsIndexField : Boolean`

Visibility: public

Access: Read

Description: `IsIndexField` is true if the field is an indexed field. By default this property is `False`, descendents of TDataset (250) can change this to `True`.

See also: TField.Calculated (309)

10.27.39 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 (265) is called or Clear (303) is called then `IsNull` will return `True`.

See also: TField.Clear (303), TDataset.Insert (265)

10.27.40 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 (316) equals `fkLookup`, `False` otherwise. Setting the `Lookup` property will switch the `FieldKind` between the `fkLookup` and `fkData`.

See also: TField.FieldKind (316)

10.27.41 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 (250) does not yet support this.

See also: TField.Value (314), TField.CurValue (310)

10.27.42 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 (311), TField.Index (317), TField.Datasize (310)

10.27.43 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 (310) should be used. For blob fields, the current size of the

10.27.44 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 (321) event handler if a handler was specified. For display purposes, the TField.DisplayText (311) 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 (308) property, this will invoke the Tfield.OnSetText (321) event handler, if one is set.

See also: TField.AsString (308), TField.DisplayText (311), TField.Value (314)

10.27.45 TField.ValidChars

Synopsis: Characters that are valid input for the field's content

Declaration: `Property ValidChars : TFieldChars`

Visibility: public

Access: Read

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 (304) function.

See also: TField.IsValidChar (304)

10.27.46 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 (312) 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 (312), TField.Text (313), TField.DisplayText (311)

10.27.47 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 (314), TField.CurValue (310), TField.NewValue (312)

10.27.48 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 (318) property is set to `True`. It can be refreshed using the RefreshLookupList (304) method.

See also: TField.RefreshLookupList (304), TField.LookupCache (318)

10.27.49 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 (311)

10.27.50 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 (315) as a message.

This feature is not yet implemented in FPC.

See also: TField.ConstraintErrorMessage (315), TField.ImportedConstraint (317)

10.27.51 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 (315) is violated.

This feature is not yet implemented in FPC.

See also: TField.CustomConstraint (315)

10.27.52 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 (256) or TDataset.Insert (265) 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 (265), TDataset.Append (256), TDataset.CustomConstraint (250)

10.27.53 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 (317) property, unless assigned another value.

See also: TField.FieldName (317)

10.27.54 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 (313) for string fields) but can be modified to obtain a more appropriate value for the field's expected content.

See also: TField.Alignment (315), TField.DisplayText (311)

10.27.55 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.

10.27.56 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 (322), TField.Size (313), TField.DataType (310)

10.27.57 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 (315) or ImportedConstraint (317) properties is set to a non-empty value.

See also: TField.CustomConstraint (315), TField.ImportedConstraint (317)

10.27.58 TField.Index

Synopsis: Index of the field in the list of fields

Declaration: `Property Index : LongInt`

Visibility: published

Access: Read,Write

Description: `Index` is the name of the field in the list of fields of a dataset. It is, in general, the (0-based) position of the field in the underlying datas structures, but this need not always be so. The TField.FieldNo (311) property should be used for that.

See also: TField.FieldNo (311)

10.27.59 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 (250) descendent.

See also: TField.CustomConstraint (315), TDataset (250), TField.ConstraintErrorMessage (315)

10.27.60 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 (319) in the dataset pointed to by the LookupDataset (318) property.

See also: TField.LookupKeyFields (319), Tfield.LookupDataset (318)

10.27.61 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 (319) in the dataset pointed to by the LookupDataset (318) property. The list of key, value pairs is available through the TField.LookupList (314) property.

See also: TField.LookupKeyFields (319), Tfield.LookupDataset (318), TField.LookupList (314)

10.27.62 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 (319) property is used as a list of fields to locate a record in this dataset, and the value of the LookupResultField (319) field is then used as the value of the lookup field.

See also: TField.KeyFields (318), TField.LookupKeyFields (319), TField.LookupResultField (319), TField.LookupCache (318)

10.27.63 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 (318) property. These fields will be used when locating a record corresponding to the values in the TField.KeyFields (318) property.

See also: TField.KeyFields (318), TField.LookupDataset (318), TField.LookupResultField (319), TField.LookupCache (318)

10.27.64 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 (318) 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: TField.KeyFields (318), TField.LookupDataset (318), TField.LookupKeyFields (319), TField.LookupCache (318)

10.27.65 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 (250) descendent or the database engine support retrieval of this property. It can be used to autmatically create update statements, together with the TField.ProviderFlags (319) property.

See also: TDataset (250), TField.ProviderFlags (319)

10.27.66 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 (319)

10.27.67 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 (325)

10.27.68 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 (250) 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 (268), TField.ReadOnly (320), TField.Visible (320)

10.27.69 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 (320), TField.Required (320)

10.27.70 TField.OnChange

Synopsis: Event triggerd 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 (322) 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 (321)

10.27.71 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 (313) or TField.DisplayText (311) 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 (313), TField.DisplayText (311), TField.OnSetText (321), TFieldGetTextEvent (214)

10.27.72 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 (313) 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 (321) handler, if it exists.

See also: TField.Text (313), TField.OnGetText (321), TFieldGetTextEvent (214)

10.27.73 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 (313), TField.OnGetText (321), TField.OnSetText (321), TField.OnChange (321)

10.28 TFieldDef

10.28.1 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.)

10.28.2 Method overview

Page	Property	Description
323	Assign	Assign the contents of one TFieldDef instance to another.
322	Create	Constructor for `TFieldDef`.
323	CreateField	Create `TField` instance based on definitions in current `TFieldDef` instance.
323	Destroy	Free the TFieldDef instance

10.28.3 Property overview

Page	Property	Access	Description
325	Attributes	rw	Additional attributes of the field.
325	DataType	rw	Data type for the field
324	FieldClass	r	TField class used for this fielddef
324	FieldNo	r	Field number
324	InternalCalcField	rw	Is this a definition of an internally calculated field ?
325	Precision	rw	Precision used in BCD (Binary Coded Decimal) fields
324	Required	rw	Is the field required ?
326	Size	rw	Size of the buffer needed to store the data of the field

10.28.4 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 (322), datatype (325), size (326), precision (325), FieldNo (324) and the Required (324) property are all set according to the passsed arguments.

Errors: If a duplicate name is passed, then an exception will occur.

See also: TFieldDef.name (322), TFieldDef.Datatype (325), TFieldDef.Size (326), TFieldDef.Precision (325), TFieldDef.FieldNo (324), TFieldDef.Required (324)

10.28.5 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 (322)

10.28.6 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 (322), then an exception will be raised.

10.28.7 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 (325) what TField (298) descendent it should create, and then returns a newly created instance of this class. It sets the appropriate defaults for the Size (313), FieldName (317), FieldNo (311), Precision (298), ReadOnly (320) and Required (320) 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 (208) 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.

Errors:

See also: DefaultFieldClasses (208), TField (298)

10.28.8 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 (323) 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 (250), TFieldDef.CreateField (323), TField (298)

10.28.9 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.

10.28.10 TFieldDef.InternalCalcField

Synopsis: Is this a definition of an internally calculated field ?

Declaration: `Property InternalCalcField : Boolean`

Visibility: public

Access: Read,Write

Description: `Internalcalc` is `True` if the fielddef instance represents an internally calculated field: for internally calculated fields, storage must be rovided by the underlying data mechanism.

10.28.11 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 excepion 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`.

10.28.12 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 (324)

10.28.13 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 mustbe used to represent this field.

See also: TFieldDef.FieldClass (324), TFieldDef.CreateField (323)

10.28.14 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 (326)

10.28.15 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 mechaism 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 (325), TFieldDef.DataType (325)

10.29 TFieldDefs

10.29.1 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.

10.29.2 Method overview

Page	Property	Description
327	Add	Add a new field definition to the collection.
327	AddFieldDef	Add new `TFieldDef`
327	Assign	Copy all items from one dataset to another
326	Create	Create a new instance of `TFieldDefs`
328	Find	Find item by name
328	MakeNameUnique	Create a unique field name starting from a base name
328	Update	Force update of definitions

10.29.3 Property overview

Page	Property	Access	Description
328	HiddenFields	rw	Should field instances be created for hidden fields
329	Items	rw	Indexed access to the fielddef instances

10.29.4 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 (322), TDataset (250)

10.29.5 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 (327)

10.29.6 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 (327)

10.29.7 TFieldDefs.Assign

Synopsis: Copy all items from one dataset to another

Declaration:
```
procedure Assign(FieldDefs: TFieldDefs);  Overload
```

Visibility: public

Description: `Assign` simply calls inherited `Assign` with the `FieldDefs` argument.

See also: TFieldDef.Assign (323)

10.29.8 TFieldDefs.Find

Synopsis: Find item by name

Declaration: `function Find(const AName: String) : TFieldDef`

Visibility: public

Description: `Find` simply calls the inherited TDefCollection.Find (296) to find an item with name `AName` and typecasts the result to `TFieldDef`.

Errors:

See also: TDefCollection.Find (296), TNamedItem.Name (354)

10.29.9 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 (297)

10.29.10 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 (328), TFieldDef.Name (322)

10.29.11 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 (325)

10.29.12 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 (322)

10.30 Tfields

10.30.1 Description

`TFields` mimics a `TCollection` class for the Fields (277) property of TDataset (250) instance. Since TField (298) is a descendent of `TComponent`, it cannot be an item of a collection, and must be managed by another class.

10.30.2 Method overview

Page	Property	Description
330	Add	Add a new field to the list
330	CheckFieldName	Check field name for duplicate entries
330	CheckFieldNames	Check a list of field names for duplicate entries
331	Clear	Clear the list of fields
329	Create	Create a new instance of `TFields`
330	Destroy	Free the `TFields` instance
331	FieldByName	Find a field based on its name
331	FieldByNumber	Search field based on its fieldnumber
331	FindField	Find a field based on its name
332	GetEnumerator	
332	GetFieldNames	Get the list of fieldnames
332	IndexOf	Return the index of a field instance
332	Remove	Remove an instance from the list

10.30.3 Property overview

Page	Property	Access	Description
333	Count	r	Number of fields in the list
333	Dataset	r	Dataset the fields belong to
333	Fields	rw	Indexed access to the fields in the list

10.30.4 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 (333) property, and initializes an internal list object to store te list of fields.

See also: TDataset (250), TFields.Dataset (333), TField (298)

10.30.5 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 (333). After that it cleans up the internal structures and then calls the inherited destructor.

See also: TDataset (250), TField (298), TFields.Clear (331)

10.30.6 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 (298) 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 (298), TFields.Clear (331)

10.30.7 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 (329)). If it does, then an EDatabaseError (220) exception is raised.

See also: TField.FieldName (317), TFields.Find (329)

10.30.8 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 (330).

Errors: Spaces are not discarded, so leaving a space after of before a fieldname will not find the fieldname, and will yield a false negative result.

See also: TField.FieldName (317), TFields.CheckFieldName (330), TFields.Find (329)

10.30.9 Tfields.Clear

Synopsis: Clear the list of fields

Declaration: `procedure Clear`

Visibility: public

Description: `Clear` removes all TField (298) var instances from the list. All field instances are freed after they have been removed from the list.

See also: TField (298)

10.30.10 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 (317) property matches `Value`. The search is performed case-insensitively. If no field instance is found, then `Nil` is returned.

See also: TFields.FieldByName (331)

10.30.11 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 (317) 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 (298), where `Nil` is returned if no match is found.

See also: TFields.FindField (331), TFields.FieldName (329), Tfields.FieldByNumber (331), TFields.IndexOf (332)

10.30.12 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 (311) property matches the `FieldNo` parameter. If no such field is found, `Nil` is returned.

See also: TFields.FieldByName (331), TFields.FindField (331), TFields.IndexOf (332)

10.30.13 Tfields.GetEnumerator

Declaration: `function GetEnumerator : TFieldsEnumerator`

Visibility: public

10.30.14 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 (317)

10.30.15 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 retuns the index of the field instance in the list (it compares actual field instances, not field names). If the field does not appear in the list, -1 is returned.

See also: TFields.FieldByName (331), TFields.FieldByNumber (331), TFields.FindField (331)

10.30.16 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: Tields.Clear (208)

10.30.17 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 (333) property are numbered from 0 to `Count-1`.

See also: TFields.fields (333)

10.30.18 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 (329) 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 (329), TField.Dataset (310), TDataset (250)

10.30.19 Tfields.Fields

Synopsis: Indexed access to the fields in the list

Declaration: `Property Fields[Index: Integer]: TField; default`

Visibility: public

Access: Read,Write

Description: `Fields` is the default property of the `TFields` class. It provides indexed access to the fields in the list: the index runs from 0 to `Count-1`.

Errors: Providing an index outside the allowed range will result in an `EListError` exception.

See also: TFields.FieldByName (331)

10.31 TFieldsEnumerator

10.31.1 Method overview

Page	Property	Description
334	Create	
334	MoveNext	

10.31.2 Property overview

Page	Property	Access	Description
334	Current	r	

10.31.3 TFieldsEnumerator.Create

Declaration: `constructor Create(AFields: TFields)`

Visibility: public

10.31.4 TFieldsEnumerator.MoveNext

Declaration: `function MoveNext : Boolean`

Visibility: public

10.31.5 TFieldsEnumerator.Current

Declaration: `Property Current : TField`

Visibility: public

Access: Read

10.32 TFloatField

10.32.1 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 (335), MaxValue (336), MinValue (336) and overrides some TField (298) 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.

10.32.2 Method overview

Page	Property	Description
335	CheckRange	Check whether a value is in the allowed range of values for the field
335	Create	Create a new instance of the `TFloatField`

10.32.3 Property overview

Page	Property	Access	Description
335	Currency	rw	Is the field a currency field.
336	MaxValue	rw	Maximum value for the field
336	MinValue	rw	Minimum value for the field
336	Precision	rw	Precision (number of digits) of the field in text representations
335	Value	rw	Value of the field as a double type

10.32.4 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.

10.32.5 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 (336) and MaxValue (336) properties. If the value lies outside of the allowed range, then `False` is returned.

See also: TFloatField.MaxValue (336), TFloatField.MinValue (336)

10.32.6 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 (307)

See also: TField.AsFloat (307), TField.Value (314)

10.32.7 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 (311) and TField.Text (313) 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 (311), TField.Text (313), TNumericField.DisplayFormat (356), TNumericField.EditFormat (356)

10.32.8 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, MinField (208) should also be set, because it will also be checked.

See also: TFloatField.MinValue (336)

10.32.9 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, MaxField (208) should also be set, because it will also be checked.

See also: TFloatField.MaxValue (336), TFloatField.CheckRange (335)

10.32.10 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 (311) or TField.Text (313), it is used in the arguments to FormatFloat (**??**).

See also: TField.Displaytext (311), TField.Text (313), #rtl.sysutils.FormatFloat (**??**)

10.33 TGraphicField

10.33.1 Description

`TGraphicField` is the class used when a dataset must manage graphical BLOB data. (TField.DataType (310) equals `ftGraphic`). It initializes some of the properties of the TField (298) class. All methods to be able to work with graphical BLOB data have been implemented in the TBlobField (227) parent class.

It should never be necessary to create an instance of `TGraphicsField` manually, a field of this class will be instantiated automatically for each graphical BLOB field when a dataset is opened.

10.33.2 Method overview

Page	Property	Description
337	Create	Create a new instance of the `TGraphicField` class

10.33.3 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 (298) properties to configure the instance for working with graphical BLOB values.

See also: TField (298)

10.34 TGuidField

10.34.1 Description

`TGUIDField` is the class used when a dataset must manage native variant-typed data. (TField.DataType (310) equals `ftGUID`). It initializes some of the properties of the TField (298) 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.

10.34.2 Method overview

Page	Property	Description
338	Create	Create a new instance of the `TGUIDField` class

10.34.3 Property overview

Page	Property	Access	Description
338	AsGuid	rw	Field content as a GUID value

10.34.4 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 (298) properties to configure the instance for working with GUID values.

See also: TField (298)

10.34.5 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 (308)

10.35 TIndexDef

10.35.1 Description

`TIndexDef` describes one index in a set of indexes of a TDataset (250) instance. The collection of indexes is descibed by the TIndexDefs (341) class. It just has the necessary properties to describe an index, but does not implement any functionality to maintain an index.

10.35.2 Method overview

Page	Property	Description
339	Create	Create a new index definition
339	Destroy	Remove the index from memory

10.35.3 Property overview

Page	Property	Access	Description
340	CaseInsFields	rw	Fields in field list that are ordered case-insensitively
340	DescFields	rw	Fields in field list that are ordered descending
339	Expression	rw	Expression that makes up the index values
339	Fields	rw	Fields making up the index
340	Options	rw	Index options
341	Source	rw	Source of the index

10.35.4 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 (338) 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 (341) 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 (341), TIndexDef.Options (340), TIndexDef.Fields (339)

10.35.5 TIndexDef.Destroy

Synopsis: Remove the index from memory

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

Visibility: public

Description: `Destroy` calls the inherited destructor, and removes the item from memory.

See also: TIndexDef.Create (339)

10.35.6 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 (340)

See also: TIndexDef.Options (340), TindexDef.Fields (339)

10.35.7 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 (340) property can be used to determine the fields in the list that have a descending sort order. The TIndexDef.CaseInsFields (340) property determines which fields are sorted in a case-insensitive manner.

See also: TIndexDef.DescFields (340), TIndexDef.CaseInsFields (340), TIndexDef.Expression (339)

10.35.8 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 (339) property which are ordered in a case-insensitive manner. `CaseInsFields` may not contain fieldnames that do not appear in `Fields`.

See also: TIndexDef.Fields (339), TIndexDef.Expression (339), TIndexDef.DescFields (340)

10.35.9 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 (339) property which are ordered in a descending manner. `DescFields` may not contain fieldnames that do not appear in `Fields`.

See also: TIndexDef.Fields (339), TIndexDef.Expression (339), TIndexDef.DescFields (340)

10.35.10 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 (216) for more information on the various available options.

See also: TIndexOptions (216)

10.35.11 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.

10.36 TIndexDefs

10.36.1 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.

10.36.2 Method overview

Page	Property	Description
342	Add	Add a new index definition with given name and options
342	AddIndexDef	Add a new, empty, index definition
341	Create	Create a new `TIndexDefs` instance
342	Destroy	Remove the indexdefs from memory.
342	Find	Find an index by name
343	FindIndexForFields	Find index definition based on field names
343	GetIndexForFields	Get index definition based on field names
343	Update	Called whenever one of the items changes

10.36.3 Property overview

Page	Property	Access	Description
343	Items	rw	Indexed access to the index definitions

10.36.4 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 (338).

See also: TIndexDef (338), TIndexDefs.Destroy (342)

10.36.5 TIndexDefs.Destroy

Synopsis: Remove the indexdefs from memory.

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

Visibility: public

Description: `Destroy` removes the indexdefs from memory. If the `TIndexDefs` instance is created by a TDataset (250) instance, the dataset instance will also destroy it.

See also: TIndexDefs.Create (341)

10.36.6 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 (338) 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 (338), TNamedItem.Name (354), TIndexDef.Fields (339), TIndexDef.Options (340), TIndexDefs.AddIndexDef (342)

10.36.7 TIndexDefs.AddIndexDef

Synopsis: Add a new, empty, index definition

Declaration:
```
function AddIndexDef : TIndexDef
```

Visibility: public

Description: `AddIndexDef` adds a new TIndexDef (338) 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 (342)

10.36.8 TIndexDefs.Find

Synopsis: Find an index by name

Declaration:
```
function Find(const IndexName: String) : TIndexDef
```

Visibility: public

Description: `Find` overloads the TDefCollection.Find (296) method to search and return a TIndexDef (338) instance based on the name. The search is case-insensitive and returns `Nil` if no matching index definition was found.

Errors:

See also: TIndexDef (338), TDefCollection.Find (296), TIndexDefs.FindIndexForFields (343)

10.36.9 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 (339) 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 (338), TIndexDefs.Find (342), TIndexDefs.GetindexForFields (343)

10.36.10 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 (339) property matches the list of fields in `Fields`. If `CaseInsenstitive` is `True` it only searches for case-sensitive indexes. If it finds an index definition, then it returns the found instance. If it does not find a matching definition, `Nil` is returned.

See also: TIndexDef (338), TIndexDefs.Find (342), TIndexDefs.FindIndexForFields (343)

10.36.11 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.

10.36.12 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 (338)

10.37 TLargeintField

10.37.1 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 (298) 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 (345) and MaxValue (208).

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.

10.37.2 Method overview

Page	Property	Description
344	CheckRange	Check whether a values falls within the allowed range
344	Create	Create a new instance of the `TLargeintField` class

10.37.3 Property overview

Page	Property	Access	Description
345	MaxValue	rw	Maximum value for the field
345	MinValue	rw	Minimum value for the field
345	Value	rw	Field contents as a 64-bit integer value

10.37.4 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 (298) and MinValue (345) and MaxValue (208).

See also: TField (298), TLargeIntField.MinValue (345), TLargeInField.MaxValue (208)

10.37.5 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 (345) and MaxValue (345) properties. If the value lies outside of the allowed range, then `False` is returned.

See also: TLargeIntField.MaxValue (345), TLargeIntField.MinValue (345)

10.37.6 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 (307).

See also: TField.Value (314), TField.AsLargeInt (307)

10.37.7 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, MinField (208) should also be set, because it will also be checked.

See also: TLargeIntField.MinValue (345)

10.37.8 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, MaxField (208) should also be set, because it will also be checked.

See also: TLargeIntField.MaxValue (345)

10.38 TLongintField

10.38.1 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 (298) 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 (347) and MaxValue (347).

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.

10.38.2 Method overview

Page	Property	Description
346	CheckRange	Check whether a valid is in the allowed range of values for the field
346	Create	Create a new instance of `TLongintField`

10.38.3 Property overview

Page	Property	Access	Description
347	MaxValue	rw	Maximum value for the field
347	MinValue	rw	Minimum value for the field
347	Value	rw	Value of the field as longint

10.38.4 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 (347) and MaxValue (347) properties.

See also: TField (298), TLongintField.MaxValue (347), TLongintField.MinValue (347)

10.38.5 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 (347) and MaxValue (347) properties. If the value lies outside of the allowed range, then `False` is returned.

See also: TLongintField.MaxValue (347), TLongintField.MinValue (347)

10.38.6 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 (308) property.

See also: TField.Value (314)

10.38.7 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: TLongintField.MinValue (347)

10.38.8 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: TLongintField.MaxValue (347)

10.39 TLookupList

10.39.1 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.

10.39.2 Method overview

Page	Property	Description
348	Add	Add a key, value pair to the list
348	Clear	Remove all key, value pairs from the list
348	Create	Create a new instance of `TLookupList`.
348	Destroy	Free a `TLookupList` instance from memory
349	FirstKeyByValue	Find the first key that matches a value
349	ValueOfKey	Look up value based on a key
349	ValuesToStrings	Convert values to stringlist

10.39.3 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 (348)

10.39.4 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: TLookpList.Create (208)

10.39.5 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 (348)

10.39.6 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 (348)

10.39.7 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 (349)

10.39.8 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 (349), TLookupList.Add (348)

10.39.9 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 (349)

10.40 TMasterDataLink

10.40.1 Description

`TMasterDataLink` is a `TDatalink` descendent which handles master-detail relations. It can be used in TDataset (250) 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 (249) property.

The class also provides functionailty for keeping a list of fields that make up the master-detail functionality, in the TMasterDatalink.FieldNames (350) and TMasterDataLink.Fields (351) properties.

This class should never be used in application code.

10.40.2 Method overview

Page	Property	Description
350	Create	Create a new instance of `TMasterDataLink`
350	Destroy	Free the datalink instance from memory

10.40.3 Property overview

Page	Property	Access	Description
350	FieldNames	rw	List of fieldnames that make up the master-detail relationship
351	Fields	r	List of fields as specified in `FieldNames`
351	OnMasterChange	rw	Called whenever the master dataset data changes
351	OnMasterDisable	rw	Called whenever the master dataset is disabled

10.40.4 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 (298) property. The master dataset must be set through the DataSource (249) property, and is usually set by the applictaion programmer.

See also: TDetailDataLink.DetailDataset (298), TDatalink.Datasource (249)

10.40.5 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 (350)

10.40.6 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 (249)) on which the master-detail relationship is based. Setting this property will fill the TMasterDataLink.Fields (351) property with the field instances of the master dataset.

See also: TMasterDataLink.Fields (351), TDatalink.Dataset (249), TDataset.GetFieldList (264)

10.40.7 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 (298) instances from the master dataset (TDatalink.Dataset (249)) when the FieldNames (350) property is set, and when the master dataset opens.

See also: TField (298), TMasterDatalink.FieldNames (350)

10.40.8 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 (351) list is empty, TMasterDataLink.OnMasterDisable (351) is called instead.

See also: TMasterDataLink.OnMasterDisable (351)

10.40.9 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 (351)

10.41 TMasterParamsDataLink

10.41.1 Description

`TMasterParamsDataLink` is a TDataLink (245) 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.

10.41.2 Method overview

Page	Property	Description
352	CopyParamsFromMaster	Copy parameter values from master dataset.
352	Create	Initialize a new `TMasterParamsDataLink` instance
352	RefreshParamNames	Refresh the list of parameter names

10.41.3 Property overview

Page	Property	Access	Description
353	Params	rw	Parameters of detail dataset.

10.41.4 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 (368) in the published properties of `ADataset` and assigns it to the Params (353) property.

See also: TDataset (250), TParams (368), TMasterParamsDataLink.Params (353)

10.41.5 TMasterParamsDataLink.RefreshParamNames

Synopsis: Refresh the list of parameter names

Declaration: `procedure RefreshParamNames; Virtual`

Visibility: public

Description: `RefreshParamNames` scans the Params (353) property and sets the FieldNames (350) property to the list of parameter names.

See also: TMasterParamsDataLink.Params (353), TMasterDataLink.FieldNames (350)

10.41.6 TMasterParamsDataLink.CopyParamsFromMaster

Synopsis: Copy parameter values from master dataset.

Declaration: `procedure CopyParamsFromMaster(CopyBound: Boolean); Virtual`

Visibility: public

Description: `CopyParamsFromMaster` calls TParams.CopyParamValuesFromDataset (371), 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 (364) 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 (371), TParam.Bound (364)

10.41.7 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 (368), TMasterParamsDataLink.Create (352)

10.42 TMemoField

10.42.1 Description

`TMemoField` is the class used when a dataset must manage memo (Text BLOB) data. (TField.DataType (310) equals `ftMemo`). It initializes some of the properties of the TField (298) class. All methods to be able to work with memo fields have been implemented in the TBlobField (227) 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.

10.42.2 Method overview

Page	Property	Description
353	Create	Create a new instance of the `TMemoField` class

10.42.3 Property overview

Page	Property	Access	Description
354	Transliterate		Should the contents of the field be transliterated

10.42.4 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 (298) properties to configure the instance for working with memo values.

See also: TField (298)

10.42.5 TMemoField.Transliterate

Synopsis: Should the contents of the field be transliterated

Declaration: `Property Transliterate :`

Visibility: published

Access:

Description: `Transliterate` is redefined from TBlobField.Transliterate (230) with a default value of `true`.

See also: TBlobField.Transliterate (230), TStringField.Transliterate (374), TDataset.Translate (270)

10.43 TNamedItem

10.43.1 Description

`NamedItem` is a TCollectionItem (**??**) descendent which introduces a Name (354) property. It automatically returns the value of the `Name` property as the value of the DisplayName (354) property.

10.43.2 Property overview

Page	Property	Access	Description
354	DisplayName	rw	Display name
354	Name	rw	Name of the item

10.43.3 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 (354) property.

See also: TNamedItem.Name (354)

10.43.4 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 vaulue for the DisplayName (354) property. If the `TNamedItem` item is owned by a TDefCollection (296) collection, then the name must be unique, i.e. each `Name` value may appear only once in the collection.

See also: TNamedItem.DisplayName (354), TDefCollection (296)

10.44 TNumericField

10.44.1 Description

`TNumericField` is an abstract class which overrides some of the methods of TField (298) 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 TNumericalField.DisplayFormat (208) and TNumericalField.EditFormat (208).

Since `TNumericalField` is an abstract class, it must never be instantiated directly. Instead one of the descendent classes should be created.

10.44.2 Method overview

Page	Property	Description
355	Create	Create a new instance of `TNumericField`

10.44.3 Property overview

Page	Property	Access	Description
355	Alignment		Alignment of the field
356	DisplayFormat	rw	Format string for display of numerical data
356	EditFormat	rw	Format string for editing of numerical data

10.44.4 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 (315) property with

See also: TField.Alignment (315)

10.44.5 TNumericField.Alignment

Synopsis: Alignment of the field

Declaration: `Property Alignment :`

Visibility: published

Access:

Description: `Alignment` is published by `TNumericalField` with `taRightJustify` as a default value.

See also: TField.Alignment (315)

10.44.6 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 (311) 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: #rtl.sysutils.Format (**??**), #rtl.sysutils.FormatFloat (**??**), TField.DisplayText (311), TNumericField.EditFormat (356)

10.44.7 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 (313) 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: #rtl.sysutils.Format (**??**), #rtl.sysutils.FormatFloat (**??**), TField.Text (313), TNumericField.DisplayFormat (356)

10.45 TParam

10.45.1 Description

`TParam` is one item in a TParams (368) collection. It describes the name (TParam.Name (366)), type (ParamType (367)) and value (TParam.Value (365)) 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 (250) descendent that owns the parameters should have created all necessary `TParam` instances.

10.45.2 Method overview

Page	Property	Description
358	Assign	Assign one parameter instance to another
358	AssignField	Copy value from field instance
359	AssignFieldValue	Assign field value to the parameter.
359	AssignFromField	Copy field type and value
358	AssignToField	Assign parameter value to field
359	Clear	Clear the parameter value
357	Create	Create a new parameter value
359	GetData	Get the parameter value from a memory buffer
360	GetDataSize	Return the size of the data.
360	LoadFromFile	Load a parameter value from file
360	LoadFromStream	Load a parameter value from stream
360	SetBlobData	Set BLOB data
361	SetData	Set the parameter value from a buffer

10.45.3 Property overview

Page	Property	Access	Description
361	AsBlob	rw	Return parameter value as a blob
361	AsBoolean	rw	Get/Set parameter value as a boolean value
361	AsCurrency	rw	Get/Set parameter value as a currency value
362	AsDate	rw	Get/Set parameter value as a date (TDateTime) value
362	AsDateTime	rw	Get/Set parameter value as a date/time (TDateTime) value
362	AsFloat	rw	Get/Set parameter value as a floating-point value
362	AsInteger	rw	Get/Set parameter value as an integer (32-bit) value
363	AsLargeInt	rw	Get/Set parameter value as a 64-bit integer value
363	AsMemo	rw	Get/Set parameter value as a memo (string) value
363	AsSmallInt	rw	Get/Set parameter value as a smallint value
363	AsString	rw	Get/Set parameter value as a string value
364	AsTime	rw	Get/Set parameter value as a time (TDateTime) value
366	AsWideString	rw	Get/Set the value as a widestring
364	AsWord	rw	Get/Set parameter value as a word value
364	Bound	rw	Is the parameter value bound (set to fixed value)
364	Dataset	r	Dataset to which this parameter belongs
366	DataType	rw	Data type of the parameter
365	IsNull	r	Is the parameter empty
366	Name	rw	Name of the parameter
365	NativeStr	rw	No description available
366	NumericScale	rw	Numeric scale
367	ParamType	rw	Type of parameter
367	Precision	rw	Precision of the BCD value
367	Size	rw	Size of the parameter
365	Text	rw	Read or write the value of the parameter as a string
365	Value	rws	Value as a variant

10.45.4 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 (368)

10.45.5 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 (298), then it is passed to TParam.AssignField (358). If `Source` is of type `TStrings`, then it is assigned to TParams.AsMemo (368).

Errors: If `Source` is not of type `TParam`, `TField` or `TStrings`, an exception will be raised.

See also: TField (298), TParam.Name (366), TParam.Bound (364), TParam.NumericScale (366), TParam.ParamType (367), TParam.value (365), TParam.Size (367), TParam.AssignField (358), Tparam.AsMemo (363)

10.45.6 TParam.AssignField

Synopsis: Copy value from field instance

Declaration: `procedure AssignField(Field: TField)`

Visibility: public

Description: `AssignField` copies the `Field`, FieldName (317) and Value (314) 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 (358), TParam.AssignToField (358), TParam.AssignFieldValue (359)

10.45.7 TParam.AssignToField

Synopsis: Assign parameter value to field

Declaration: `procedure AssignToField(Field: TField)`

Visibility: public

Description: `AssignToField` copies the parameter value (365) to the `field` instance. If `Field` is `Nil`, nothing happens.

Errors: An EDatabaseError (220) exception is raised if the field has an unsupported field type (for types ftCursor, ftArray, ftDataset,ftReference).

See also: TParam.Assign (358), TParam.AssignField (358), TParam.AssignFromField (359)

10.45.8 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 (364) bound parameter to `True`. This method is called from TParam.AssignField (358).

See also: TField (298), TParam.AssignField (358), TParam.Bound (364)

10.45.9 TParam.AssignFromField

Synopsis: Copy field type and value

Declaration: `procedure AssignFromField(Field: TField)`

Visibility: public

Description: `AssignFromField` copies the field value (314) and data type (TField.DataType (310)) to the parameter instance. If `Field` is `Nil`, nothing happens. This is the reverse operation of TParam.AssignToField (358).

Errors: An EDatabaseError (220) exception is raised if the field has an unsupported field type (for types ftCursor, ftArray, ftDataset,ftReference).

See also: TParam.Assign (358), TParam.AssignField (358), TParam.AssignToField (358)

10.45.10 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 (365), TParam.Name (366), TParam.ParamType (367), TParam.DataType (366)

10.45.11 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 (298), 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 (220) exception.

See also: TParam.SetData (361), TField (298)

10.45.12 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 (220) exception is raised when this function is called.

See also: TParam.GetData (359), TParam.SetData (361)

10.45.13 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 (360).

Errors: If the specified `FileName` is not a valid file, or the file is not readable, an exception will occur.

See also: TParam.LoadFromStream (360), TBlobType (210), TParam.SaveToFile (356)

10.45.14 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 (360), TParam.SaveToStream (356)

10.45.15 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 (360)

10.45.16 TParam.SetData

Synopsis: Set the parameter value from a buffer

Declaration: `procedure SetData(Buffer: Pointer)`

Visibility: public

Description: `SetData` performs the rever operation of TParam.GetData (359): it reads the parameter value from the memory area pointed to by `Buffer`. The size of the data read is determined by TParam.GetDataSize (360) and the type of data by TParam.DataType (366) : it is the same storage mechanism used by TField (298), and so can be used to copy the value from a TDataset (250) record buffer.

Errors: Not all field types are supported. If an unsupported field type is encountered, an EDatabaseError (220) exception is raised.

See also: TDataset (250), TParam.GetData (359), TParam.DataType (366), TParam.GetDataSize (360)

10.45.17 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 (363)

10.45.18 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 (366), TParam.Value (365)

10.45.19 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 (362), TParam.DataType (366), TParam.Value (365)

10.45.20 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 (362), TParam.AsTime (364), TParam.DataType (366), TParam.Value (365)

10.45.21 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 (362), TParam.asTime (364), TParam.DataType (366), TParam.Value (365)

10.45.22 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 (361), TParam.DataType (366), TParam.Value (365)

10.45.23 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 (363), TParam.AsSmallInt (363), TParam.AsWord (364), TParam.DataType (366), TParam.Value (365)

10.45.24 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 (362), TParam.asSmallint (363), TParam.AsWord (364), TParam.DataType (366), TParam.Value (365)

10.45.25 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 (363), TParam.LoadFromStream (360), TParam.SaveToStream (356), TParam.DataType (366), TParam.Value (365)

10.45.26 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 (362), TParam.AsLargeInt (363), TParam.AsWord (364), TParam.DataType (366), TParam.Value (365)

10.45.27 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 (366), TParam.Value (365)

10.45.28 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 (362), TParam.AsDateTime (362), TParam.DataType (366), TParam.Value (365)

10.45.29 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 (362), TParam.AsLargeInt (363), TParam.AsSmallint (363), TParam.DataType (366), TParam.Value (365)

10.45.30 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 (366), TParam.Value (365)

10.45.31 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 (368) 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 (250), TParams (368)

10.45.32 TParam.IsNull

Synopsis: Is the parameter empty

Declaration: `Property IsNull : Boolean`

Visibility: public

Access: Read

Description: `IsNull` is `True` is the value is empty or not set (Null or UnAssigned).

See also: TParam.Clear (359), TParam.Value (365)

10.45.33 TParam.NativeStr

Synopsis: No description available

Declaration: `Property NativeStr : String`

Visibility: public

Access: Read,Write

Description: No description available

10.45.34 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 (363), but, when written, does not set the data type: instead, it attempts to convert the value to the type specified in TParam.Datatype (366).

See also: TParam.AsString (363), TParam.DataType (366)

10.45.35 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 (366)

10.45.36 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 (366) to `ftWideString`.

See also: TParam.AsString (363), TParam.Value (365), TParam.DataType (366)

10.45.37 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 (365), TParam.Value (365), TParam.AssignField (358)

10.45.38 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 (366), TParam.Value (365), TParams.ParamByName (370)

10.45.39 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 (367), TParam.Size (367)

10.45.40 TParam.ParamType

Synopsis: Type of parameter

Declaration: `Property ParamType : TParamType`

Visibility: published

Access: Read,Write

Description: `ParamTyp` specifies the type of parameter: is the parameter value written to the database engine, or is it received from the database engine, or both ? It can have the following value:

ptUnknownUnknown type

ptInputInput parameter

ptOutputOutput 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 (366), TParam.DataSize (356), TParam.Name (366)

10.45.41 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 (366), TParam.Size (367)

10.45.42 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 (310) in the TParam.AssignFieldValue (359) method. The actual size can be retrieved through the TParam.Datasize (356) property.

See also: TParam.Datasize (356), TField.DataSize (310), TParam.AssignFieldValue (359)

10.46 TParams

10.46.1 Description

`TParams` is a collection of TParam (356) values. It is used to specify parameter values for parametrized SQL statemens, but is also used to specify parameter values for stored procedures. Its default property is an array of TParam (356) 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 (370).

TDataset (250) itself does not use `TParams`. The class is provided in the DB unit, so all `TDataset` descendents that need some kind of parametrization make use of the same interface. The TMasterParamsDataLink (351) 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 (371) method exists.

10.46.2 Method overview

Page	Property	Description
369	AddParam	Add a parameter to the collection
369	AssignValues	Copy values from another collection
371	CopyParamValuesFromDataset	Copy parameter values from a the fields in a dataset.
368	Create	Create a new instance of `TParams`
369	CreateParam	Create and add a new parameter to the collection
369	FindParam	Find a parameter with given name
370	GetParamList	Fetch a list of `TParam` instances
370	IsEqual	Is the list of parameters equal
370	ParamByName	Return a parameter by name
370	ParseSQL	Parse SQL statement, replacing parameter names with SQL parameter placeholders
371	RemoveParam	Remove a parameter from the collection

10.46.3 Property overview

Page	Property	Access	Description
372	Dataset	r	Dataset that owns the `TParams` instance
372	Items	rw	Indexed access to `TParams` instances in the collection
372	ParamValues	rw	Named access to the parameter values.

10.46.4 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 (356) 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.collection.create (**??**), TParam (356)

10.46.5 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 (356), #rtl.classes.tcollection.add (**??**)

10.46.6 TParams.AssignValues

Synopsis: Copy values from another collection

Declaration: `procedure AssignValues(Value: TParams)`

Visibility: public

Description: `AssignValues` examines all TParam (356) 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 (358)). If it is not found, nothing is done.

See also: TParam (356), TParam.Assign (358)

10.46.7 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 (356) 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 (356), TParam.Name (366), TParam.Datatype (366), TParam.Paramtype (367)

10.46.8 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 (356) 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 (366), TParams.ParamByName (370), TParams.GetParamList (370)

10.46.9 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 (356), TParam.Name (366), TParams.ParamByName (370)

10.46.10 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 (356)

10.46.11 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 (356) 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 (366), TParams.FindParam (369), TParams.GetParamList (370)

10.46.12 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 (356), TParam.Name (366), TParamStyle (217)

10.46.13 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 (356)

10.46.14 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 (358). By default, this operation is only performed on `TParam` instances with their Bound (364) 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 (356), TParam.Bound (364), TParam.AssignField (358), TDataset (250), TDataset.FieldByName (261)

10.46.15 TParams.Dataset

Synopsis: Dataset that owns the `TParams` instance

Declaration: `Property Dataset : TDataSet`

Visibility: public

Access: Read

Description: `Dataset` is the TDataset (250) instance that was specified when the `TParams` instance was created.

See also: TParams.Create (368), TDataset (250)

10.46.16 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: TParame (208)

10.46.17 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 (365)) by name. It is equivalent to reading and writing

```
ParamByName(ParamName).Value
```

See also: TParam.Value (365), TParams.ParamByName (370)

10.47 TSmallintField

10.47.1 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.

10.47.2 Method overview

Page	Property	Description
373	Create	Create a new instance of the `TSmallintField` class.

10.47.3 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 (373) class. It calls the inherited constructor and then simply sets some of the TField (298) properties to work with 16-bit signed integer data.

See also: TField (298)

10.48 TStringField

10.48.1 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 (298) 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.

10.48.2 Method overview

Page	Property	Description
374	Create	Create a new instance of the TStringField class
374	SetFieldType	

10.48.3 Property overview

Page	Property	Access	Description
374	FixedChar	rw	Is the string declared with a fixed length ?
375	Size		Maximum size of the string
374	Transliterate	rw	Should the field value be transliterated when reading or writing
374	Value	rw	Value of the field as a string

10.48.4 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 (298) properties after having called the inherited constructor.

10.48.5 TStringField.SetFieldType

Declaration: `procedure SetFieldType(AValue: TFieldType);  Override`

Visibility: public

10.48.6 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 obain 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.

10.48.7 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 (270).

See also: TDataset.Translate (270)

10.48.8 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 (308) when read, or sets the `AsString` property when written to.

See also: TField.AsString (308), TField.Value (314)

10.48.9 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 (313)

10.49 TTimeField

10.49.1 Description

`TimeField` is the class used when a dataset must manage data of type time. (TField.DataType (310) equals `ftTime`). It initializes some of the properties of the TField (298) 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.

10.49.2 Method overview

Page	Property	Description
375	Create	Create a new instance of a `TTimeField` class.

10.49.3 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 (298) properties to configure the instance for working with time values.

See also: TField (298)

10.50 TVarBytesField

10.50.1 Description

`TVarBytesField` is the class used when a dataset must manage data of variable-size binary type. (TField.DataType (310) equals `ftVarBytes`). It initializes some of the properties of the TField (298) 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.

10.50.2 Method overview

Page	Property	Description
376	Create	Create a new instance of a TVarBytesField class.

10.50.3 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 (298) properties to configure the instance for working with variable-size binary data values.

See also: TField (298)

10.51 TVariantField

10.51.1 Description

TVariantField is the class used when a dataset must manage native variant-typed data. (TField.DataType (310) equals ftVariant). It initializes some of the properties of the TField (298) 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.

10.51.2 Method overview

Page	Property	Description
376	Create	Create a new instance of the TVariantField class

10.51.3 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 (298) properties to configure the instance for working with variant values.

See also: TField (298)

10.52 TWideMemoField

10.52.1 Description

`TWideMemoField` is the class used when a dataset must manage memo (Text BLOB) data. (TField.DataType (310) equals `ftWideMemo`). It initializes some of the properties of the TField (298) class. All methods to be able to work with widestring memo fields have been implemented in the TBlobField (227) 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.

10.52.2 Method overview

Page	Property	Description
377	Create	Create a new instance of the `TWideMemoField` class

10.52.3 Property overview

Page	Property	Access	Description
377	Value	rw	Value of the field's contents as a widestring

10.52.4 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 (298) properties to configure the instance for working with widestring memo values.

See also: TField (298)

10.52.5 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 (308) property.

See also: TField.Value (314), Tfield.AsWideString (308)

10.53 TWideStringField

10.53.1 Description

`TWideStringField` is the string field class instantiated for fields of data type `ftWideString`. This class overrides some of the standard TField (298) 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.

10.53.2 Method overview

Page	Property	Description
378	Create	Create a new instance of the `TWideStringField` class.

10.53.3 Property overview

Page	Property	Access	Description
378	Value	rw	Value of the field as a widestring

10.53.4 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 (298) properties after having called the inherited constructor.

10.53.5 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 (308) property.

See also: TField.AsWideString (308), TField.Value (314)

10.54 TWordField

10.54.1 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.

10.54.2 Method overview

Page	Property	Description
379	Create	Create a new instance of the `TWordField` class.

10.54.3 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 (378) class. It calls the inherited constructor and then simply sets some of the TField (298) properties to work with 16-bit unsigned integer data.

See also: TField (298)

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 (unytped)
ftCurrency	Currency value (4 decimal points)
ftCursor	Cursor data value (no size)
ftDataSet	Dataset data (blob)
ftDate	Date value
ftDateTime	Date/Time (timestamp) value
ftDBaseOle	Paradox OLE field data
ftFixedChar	Fixed character array (string)
ftFixedWideChar	Fixed wide character date (2 bytes per character)
ftFloat	Floating point value (double)
ftFMTBcd	Formatted BCD (Binary Coded Decimal) value.
ftFmtMemo	Formatted memo ata value (no size)
ftGraphic	Graphical data value (no size)
ftGuid	GUID data value
ftIDispatch	Dispatch data value
ftInteger	Regular integer value (4 bytes, signed)
ftInterface	interface data value
ftLargeint	Large integer value (8-byte)
ftMemo	Binary text data (no size)
ftOraBlob	Oracle BLOB data
ftOraClob	Oracle CLOB data
ftParadoxOle	Paradox OLE field data (no size)
ftReference	Reference data
ftSmallint	Small integer value(1 byte, signed)
ftString	String data value (ansistring)
ftTime	Time value
ftTimeStamp	Timestamp data value
ftTypedBinary	Binary typed data (no size)
ftUnknown	Unknown data type
ftVarBytes	Array of bytes value, variable size (untyped)
ftVariant	Variant data value
ftWideMemo	Widestring memo data
ftWideString	Widestring (2 bytes per character)
ftWord	Word-sized value(2 bytes, unsigned)

Table 10.10: Enumeration values for type TFilterOption

Value	Explanation
foCaseInsensitive	Filter case insensitively.
foNoPartialCompare	Do not compare values partially, always compare completely.

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.

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 succesfully

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

Table 10.14: Enumeration values for type TLocateOption

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

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.

Table 10.16: Enumeration values for type TParamType

Value	Explanation
ptInput	Input parameter
ptInputOutput	Input/output parameter
ptOutput	Output paramete, filled on result
ptResult	Result parameter
ptUnknown	Unknown type

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`.

Table 10.18: 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

Table 10.19: 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)

Table 10.20: 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.

Table 10.21: 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.

Table 10.22: 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

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 transparant 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 (384) 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 (384) 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 (384) 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 (384) 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 (384) unit should also be functional.

11.3 Constants, types and variables

11.3.1 Resource strings

```
SEntering = '> Entering '
```

String used when sending method enter message.

```
SExiting = '< Exiting '
```

String used when sending method exit message.

```
SProcessID = 'Process %s'
```

String used when sending identification message to the server.

```
SSeparator = '>-=-=-=-=-=-=-=-=-=-=-=-=-=-=-<'
```

String used when sending a separator line.

```
SServerStartFailed = 'Failed to start debugserver. (%s)'
```

String used to display an error message when the start of the debug server failed

11.3.2 Constants

```
SendError : String = ''
```

Whenever a call encounteres an exception, the exception message is stored in this variable.

11.3.3 Types

```
TDebugLevel = (dlInformation,dlWarning,dlError)
```

Table 11.1: Enumeration values for type TDebugLevel

Value	Explanation
dlError	Error message
dlInformation	Informational message
dlWarning	Warning message

`TDebugLevel` indicates the severity level of the debug message to be sent. By default, an informational message is sent.

11.4 Procedures and functions

11.4.1 GetDebuggingEnabled

Synopsis: Check if sending of debug messages is enabled.

Declaration: `function GetDebuggingEnabled : Boolean`

Visibility: default

Description: `GetDebuggingEnabled` returns the value set by the last call to `SetDebuggingEnabled`. It is `True` by default.

See also: SetDebuggingEnabled (389), SendDebug (386)

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 (386) call will initialize the debug client when it is first called.

Errors: None.

See also: SendDebug (386), StartDebugServer (389)

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 (386) which sends the name and value of a boolean value as an informational message.

Errors: None.

See also: SendDebug (386), SendDateTime (386), SendInteger (388), SendPointer (389)

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 (386) 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 (386), SendBoolean (386), SendInteger (388), SendPointer (389)

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 (385).

See also: SendDebugEx (387), SendDebugFmt (387), SendDebugFmtEx (387)

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 (386) uses informational messages.

Other than that the function of `SendDebugEx` is equal to that of `SendDebug`

Errors: None.

See also: SendDebug (386), SendDebugFmt (387), SendDebugFmtEx (387)

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 (386). It exists mainly to avoid the `Format` call in calling code.

Errors: None.

See also: SendDebug (386), SendDebugEx (387), SendDebugFmtEx (387), #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 (387) with Debug level `MType`. It exists mainly to avoid the `Format` call in calling code.

Errors: None.

See also: SendDebug (386), SendDebugEx (387), SendDebugFmt (387), #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 (386) 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 (386), SendBoolean (386), SendDateTime (386), SendPointer (389)

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 (388), 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 (386), SendMethodExit (388), SendSeparator (389)

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 (388), 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 (386), SendMethodEnter (388), SendSeparator (389)

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 (386) 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 (386), SendBoolean (386), SendDateTime (386), SendInteger (388)

11.4.13 SendSeparator

Synopsis: Send a separator message

Declaration: `procedure SendSeparator`

Visibility: default

Description: `SendSeparator` is a simple wrapper around SendDebug (386) 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 (386), SendMethodEnter (388), SendMethodExit (388)

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 (386) 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 (385), SendDebug (386)

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 (386) call will attempt to start the server by itself if it is not yet running.

Errors: On error, `False` is returned.

See also: SendDebug (386), InitDebugClient (386)

Chapter 12

Reference for unit 'dbugmsg'

12.1 Used units

Table 12.1: Used units by unit 'dbugmsg'

Name	Page
Classes	??

12.2 Overview

dbugmsg is an auxialiary unit used in the dbugintf (384) 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 (393) and ReadDebugMessageFromStream (392) 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 (392) 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 (392) 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 (392), WriteDebugMessageToStream (393)

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 (392) 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 (392), ReadDebugMessageToStream (391)

Chapter 13

Reference for unit 'eventlog'

13.1 Used units

Table 13.1: Used units by unit 'eventlog'

Name	Page
Classes	??
sysutils	??

13.2 Overview

The EventLog unit implements the TEventLog (395) 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 (402) event handler. It should return a OS event catagory 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 (402) and OnGetCustomEventID (402) 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 (399) 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 (395) component to indicate errors.

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 (398) call, or the specialized Warning (398), Error (398), Info (399) or Debug (399) calls, which have the event type predefined.

13.5.2 Method overview

Page	Property	Description
399	Debug	Log a debug message
396	Destroy	Clean up `TEventLog` instance
398	Error	Log an error message to
396	EventTypeToString	Create a string representation of an event type
399	Info	Log an informational message
398	Log	Log a message to the system log.
397	RegisterMessageFile	Register message file
398	Warning	Log a warning message.

13.5.3 Property overview

Page	Property	Access	Description
400	Active	rw	Activate the log mechanism
401	CustomLogType	rw	Custom log type ID
400	DefaultEventType	rw	Default event type for the Log (398) call.
401	EventIDOffset	rw	Offset for event ID messages identifiers
400	FileName	rw	File name for log file
399	Identification	rw	Identification string for messages
399	LogType	rw	Log type
402	OnGetCustomCategory	rw	Event to retrieve custom message category
402	OnGetCustomEvent	rw	Event to retrieve custom event Code
402	OnGetCustomEventID	rw	Event to retrieve custom event ID
400	RaiseExceptionOnError	rw	Determines whether logging errors are reported or ignored
401	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 (400) property to False.

See also: TEventLog.Active (400)

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: TEventType (394)

13.5.6 TEventLog.RegisterMessageFile

Synopsis: Register message file

Declaration: `function RegisterMessageFile(AFileName: String) : Boolean; Virtual`

Visibility: public

Description: `RegisterMessageFile` is used on Windows to register the file `AFileName` containing the formatting strings for the system messages. This should be a file containing resource strings. If `AFileName` is empty, the filename of the application binary is substituted.

When a message is logged to the windows system log, Windows looks for a formatting string in the file registered with this call.

There are 2 kinds of formatting strings:

Category strings these should be numbered from 1 to 4

1Should contain the description of the `etInfo` event type.

2Should contain the description of the `etWarning` event type.

4Should contain the description of the `etError` event type.

4Should contain the description of the `etDebug` event type.

None of these strings should have a string substitution placeholder.

The second type of strings are the **message definitions**. Their number starts at EventIDOffset (401) (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.Log

Synopsis: Log a message to the system log.

Declaration:
```
procedure Log(EventType: TEventType;Msg: String);  Overload
procedure Log(EventType: TEventType;Fmt: String;Args: Array of const)
            ;  Overload
procedure Log(Msg: String);  Overload
procedure Log(Fmt: String;Args: Array of const);  Overload
```

Visibility: public

Description: `Log` sends a log message to the system log. The message is either the parameter `Msg` as is, or is formatted from the `Fmt` and `Args` parameters. If `EventType` is specified, then it is used as the message event type. If `EventType` is omitted, then the event type is determined from DefaultEventType (400).

If `EventType` is `etCustom`, then the OnGetCustomEvent (402), OnGetCustomEventID (402) and OnGetCustomCategory (402).

The other logging calls: Info (399), Warning (398), Error (398) and Debug (399) use the `Log` call to do the actual work.

See also: TEventLog.Info (399), TEventLog.Warning (398), TEventLog.Error (398), TEventLog.Debug (399), TEventLog.OnGetCustomEvent (402), TEventLog.OnGetCustomEventID (402), TEventLog.OnGetCustomCategory (402)

13.5.8 TEventLog.Warning

Synopsis: Log a warning message.

Declaration:
```
procedure Warning(Msg: String);  Overload
procedure Warning(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: TEventLog.Log (398), TEventLog.Info (399), TEventLog.Error (398), TEventLog.Debug (399)

13.5.9 TEventLog.Error

Synopsis: Log an error message to

Declaration:
```
procedure Error(Msg: String);  Overload
procedure Error(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: TEventLog.Log (398), TEventLog.Info (399), TEventLog.Warning (398), TEventLog.Debug (399)

13.5.10 TEventLog.Debug

Synopsis: Log a debug message

Declaration:
```
procedure Debug(Msg: String);  Overload
procedure Debug(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: TEventLog.Log (398), TEventLog.Info (399), TEventLog.Warning (398), TEventLog.Error (398)

13.5.11 TEventLog.Info

Synopsis: Log an informational message

Declaration:
```
procedure Info(Msg: String);  Overload
procedure Info(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: TEventLog.Log (398), TEventLog.Warning (398), TEventLog.Error (398), TEventLog.Debug (399)

13.5.12 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: TEventLog.Active (400), TEventLog.TimeStampFormat (401)

13.5.13 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 (400) property.

See also: TEventLog.FileName (400)

13.5.14 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: TEventLog.Log (398)

13.5.15 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.

13.5.16 TEventLog.DefaultEventType

Synopsis: Default event type for the Log (398) call.

Declaration: `Property DefaultEventType : TEventType`

Visibility: published

Access: Read,Write

Description: `DefaultEventType` is the event type used by the Log (398) call if it's `EventType` parameter is omitted.

See also: TEventLog.Log (398)

13.5.17 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 (394) 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: TEventType.LogType (394)

13.5.18 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 (399)

13.5.19 TEventLog.CustomLogType

Synopsis: Custom log type ID

Declaration: `Property CustomLogType : Word`

Visibility: published

Access: Read,Write

Description: `CustomLogType` is used in the EventTypeToString (396) to format the custom log event type string.

See also: TEventLog.EventTypeToString (396)

13.5.20 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 (397) 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: TEventLog.RegisterMessageFile (397)

13.5.21 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: TEventLog.OnGetCustomEventID (402), TEventLog.OnGetCustomEvent (402)

13.5.22 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: TEventLog.OnGetCustomCategory (402), TEventLog.OnGetCustomEvent (402)

13.5.23 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: TEventLog.OnGetCustomCategory (402), TEventLog.OnGetCustomEventID (402)

Chapter 14

Reference for unit 'ezcgi'

14.1 Used units

Table 14.1: Used units by unit 'ezcgi'

Name	Page
Classes	??
strings	403
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 (406) or DoGet (406) methods should be overidden.

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 (404)

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 (406) or DoGet (406) methods. To run the program, an instance of this class must be created, and it's Run (405) 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
404	Create	Creates a new instance of the `TEZCGI` component
404	Destroy	Removes the `TEZCGI` component from memory
406	DoGet	Method to handle `GET` requests
406	DoPost	Method to handle `POST` requests
406	GetValue	Return the value of a request variable.
405	PutLine	Send a line of output to the web-client
405	Run	Run the CGI application.
405	WriteContent	Writes the content type to standard output

14.5.3 Property overview

Page	Property	Access	Description
408	Email	rw	Email of the server administrator
408	Name	rw	Name of the server administrator
407	Names	r	Indexed array with available variable names.
406	Values	r	Variables passed to the CGI script
408	VariableCount	r	Number of available variables.
407	Variables	r	Indexed array with variables as name=value pairs.

14.5.4 TEZcgi.Create

Synopsis: Creates a new instance of the `TEZCGI` component

Declaration: `constructor Create`

Visibility: public

Description: `Create` initializes the CGI program's environment: it reads the environment variables passed to the CGI program and stores them in the Variable (403) property.

See also: TZECGI.Variables (403), TZECGI.Names (403), TZECGI.Values (403)

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: TEZcgi.Create (404)

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 (403) property) and calls the DoPost (406) or DoGet (406) 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: TZECGI.Variables (403), TEZCGI.DoPost (406), TEZCGI.DoGet (406)

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 (406) or DoGet (406) methods.

See also: TEZCGI.DoPost (406), TEZCGI.DoGet (406), TEZcgi.PutLine (405)

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 (405) 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: TEZcgi.WriteContent (405)

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: TEZCGI.Values (406)

14.5.10 TEZcgi.DoPost

Synopsis: Method to handle `POST` requests

Declaration: `procedure DoPost;  Virtual`

Visibility: public

Description: `DoPost` is called by the Run (405) 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: TEZcgi.Run (405), TEZcgi.DoGet (406)

14.5.11 TEZcgi.DoGet

Synopsis: Method to handle `GET` requests

Declaration: `procedure DoGet;  Virtual`

Visibility: public

Description: `DoGet` is called by the Run (405) 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: TEZcgi.Run (405), TEZcgi.DoPost (406)

14.5.12 TEZcgi.Values

Synopsis: Variables passed to the CGI script

Declaration: `Property Values[Index: String]: String`

Visibility: public

Access: Read

Description: `Values` is a name-based array of variables that were passed to the script by the web server or the HTTP request. The `Index` variable is the name of the variable whose value should be retrieved. The following standard values are available:

AUTH_TYPE Authorization type

CONTENT_LENGTH Content length

CONTENT_TYPE Content type

GATEWAY_INTERFACE Used gateway interface

PATH_INFO Requested URL

PATH_TRANSLATED Transformed URL

QUERY_STRING Client query string

REMOTE_ADDR Address of remote client

REMOTE_HOSTDNS name of remote client

REMOTE_IDENT Remote identity.

REMOTE_USER Remote user

REQUEST_METHOD Request methods (POST or GET)

SCRIPT_NAME Script name

SERVER_NAME Server host name

SERVER_PORT Server port

SERVER_PROTOCOL Server protocol

SERVER_SOFTWARE Web server software

HTTP_ACCEPT Accepted responses

HTTP_ACCEPT_CHARSET Accepted character sets

HTTP_ACCEPT_ENCODING Accepted encodings

HTTP_IF_MODIFIED_SINCE Proxy information

HTTP_REFERER Referring page

HTTP_USER_AGENT Client software name

Other than the standard list, any variables that were passed by the web-client request, are also available. Note that the variables are case insensitive.

See also: TEZCGI.Variables (407), TEZCGI.Names (407), TEZCGI.GetValue (406), TEZcgi.VariableCount (408)

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 (408). Any other value will result in an exception being raised.

See also: TEZcgi.Variables (407), TEZcgi.Values (406), TEZcgi.GetValue (406), TEZcgi.VariableCount (408)

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 (408). Any other value will result in an exception being raised.

See also: TEZcgi.Names (407), TEZcgi.Values (406), TEZcgi.GetValue (406), TEZcgi.VariableCount (408)

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 (407) and Variables (407) properties.

See also: TEZcgi.Names (407), TEZcgi.Variables (407), TEZcgi.Values (406), TEZcgi.GetValue (406)

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 (405) method.

See also: TEZcgi.Run (405), TEZcgi.Email (408)

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 (405) method.

See also: TEZcgi.Run (405), TEZcgi.Name (408)

Chapter 15

Reference for unit 'fpTimer'

15.1 Used units

Table 15.1: Used units by unit 'fpTimer'

Name	Page
Classes	??

15.2 Overview

The fpTimer unit implements a timer class TFPTimer (411) which can be used on all supported platforms. The timer class uses a driver class TFPTimerDriver (412) 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 (409) variable to the class pointer of the driver class. The driver class should descend from TFPTimerDriver (412).

15.3 Constants, types and variables

15.3.1 Types

```
TFPTimerDriverClass = Class of TFPTimerDriver
```

`TFPTimerDriverClass` is the class pointer of TFPTimerDriver (412) it exists mainly for the purpose of being able to set DefaultTimerDriverClass (409), so a custom timer driver can be used for the timer instances.

15.3.2 Variables

```
DefaultTimerDriverClass : TFPTimerDriverClass = nil
```

DefaultTimerDriverClass contains the TFPTimerDriver (412) class pointer that should be used when a new instance of TFPCustomTimer (410) 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 (412)) to do the actual work.

TFPCustomTimer publishes no events or properties, so it is unsuitable for handling in an IDE. The TFPTimer (411) descendent class publishes all needed events of TFPCustomTimer.

15.4.2 Method overview

Page	Property	Description
410	Create	Create a new timer
410	Destroy	Release a timer instance from memory
411	StartTimer	Start the timer
411	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 (410)

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 (410)

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 (411) event is called.

See also: TFPCustomTimer.StopTimer (411), TFPTimer.Enabled (411), TFPTimer.OnTimer (412)

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: TFPCustomTimer.StartTimer (411), TFPTimer.Enabled (411), TFPTimer.OnTimer (412)

15.5 TFPTimer

15.5.1 Description

`TFPTimer` implements no new events or properties, but merely publishes events and properties already implemented in TFPCustomTimer (410): Enabled (411), OnTimer (412) and Interval (412).

The `TFPTimer` class is suitable for use in an IDE.

15.5.2 Property overview

Page	Property	Access	Description
411	Enabled		Start or stop the timer
412	Interval		Timer tick interval in milliseconds.
412	OnTimer		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 (411)), setting it to `False` will stop the timer (calling StopTimer (411)).

See also: TFPCustomTimer.StartTimer (411), TFPCustomTimer.StopTimer (411), TFPTimer.OnTimer (412), TFPTimer.Interval (412)

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 (412) 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: TFPTimer.OnTimer (412), TFPTimer.Enabled (411)

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: TFPTimer.Interval (412), TFPTimer.Enabled (411)

15.6 TFPTimerDriver

15.6.1 Description

`TFPTimerDriver` is the abstract timer driver class: it simply provides an interface for the TFPCustomTimer (410) class to use.

The `fpTimer` unit implements a descendent of this class which implements the default timer mechanism.

15.6.2 Method overview

Page	Property	Description
413	Create	Create new driver instance
413	StartTimer	Start the timer
413	StopTimer	Stop the timer

15.6.3 Property overview

Page	Property	Access	Description
413	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 (413)) a reference to the `ATimer` instance which created the driver instance.

See also: TFPTimerDriver.Timer (413), TFPTimer (411)

15.6.5 TFPTimerDriver.StartTimer

Synopsis: Start the timer

Declaration: `procedure StartTimer; Virtual; Abstract`

Visibility: public

Description: `StartTimer` is called by TFPCustomTimer.StartTimer (411). It should be overridden by descendents of `TFPTimerDriver` to actually start the timer.

See also: TFPCustomTimer.StartTimer (411), TFPTimerDriver.StopTimer (413)

15.6.6 TFPTimerDriver.StopTimer

Synopsis: Stop the timer

Declaration: `procedure StopTimer; Virtual; Abstract`

Visibility: public

Description: `StopTimer` is called by TFPCustomTimer.StopTimer (411). It should be overridden by descendents of `TFPTimerDriver` to actually stop the timer.

See also: TFPCustomTimer.StopTimer (411), TFPTimerDriver.StartTimer (413)

15.6.7 TFPTimerDriver.Timer

Synopsis: Timer tick

Declaration: `Property Timer : TFPCustomTimer`

Visibility: public

Access: Read

Description: `Timer` calls the TFPCustomTimer (410) timer event. Descendents of `TFPTimerDriver` should call `Timer` whenever a timer tick occurs.

See also: TFPTimer.OnTimer (412), TFPTimerDriver.StartTimer (413), TFPTimerDriver.StopTimer (413)

Chapter 16

Reference for unit 'gettext'

16.1 Used units

Table 16.1: Used units by unit 'gettext'

Name	Page
Classes	??
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 (416)) 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 (415) array.

```
PMOStringTable = ^TMOStringTable
```

Pointer to a TMOStringTable (415) array.

```
PPCharArray = ^TPCharArray
```

Pointer to a TPCharArray (415) array.

```
TLongWordArray = Array[0..(1shl30)divSizeOf(LongWord)] of LongWord
```

`TLongWordArray` is an array used to define the PLongWordArray (414) 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 (415) record at the start of the file.

```
TMOStringTable = Array[0..(1shl30)divSizeOf(TMOStringInfo)] of TMOStringInfo
```

`TMOStringTable` is an array type containing TMOStringInfo (415) records. It should never be used directly, as it would occupy too much memory.

```
TPCharArray = Array[0..(1shl30)divSizeOf(PChar)] of PChar
```

`TLongWordArray` is an array used to define the PPCharArray (415) 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 (416) 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 (416), TMOFile (416)

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 (416), 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 (416), TMOFile (416)

16.5 EMOFileError

16.5.1 Description

`EMOFileError` is raised in case an TMOFile (416) instance is created with an invalid .mo.

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
417	Create	Create a new instance of the `TMOFile` class.
417	Destroy	Removes the `TMOFile` instance from memory
417	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 (415) structure, then an EMOFileError (416) exception is raised.

See also: TMOFile.Destroy (417)

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 (417)

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 'idea'

17.1 Used units

Table 17.1: Used units by unit 'idea'

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

17.2 Overview

Besides some low level IDEA encryption routines, the IDEA unit also offers 2 streams which offer on-the-fly encryption or decryption: there are 2 stream objects: A write-only encryption stream which encrypts anything that is written to it, and a decription stream which decrypts anything that is read from it.

17.3 Constants, types and variables

17.3.1 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

17.3.2 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 (420) or DeKeyIDEA (419) 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.

17.4 Procedures and functions

17.4.1 CipherIdea

Synopsis: Encrypt or decrypt a buffer.

Declaration:
```
procedure CipherIdea(Input: TIdeaCryptData;var 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 (420), DeKeyIdea (419), TIDEAEncryptStream (421), TIDEADecryptStream (420)

17.4.2 DeKeyIdea

Synopsis: Create a decryption key from an encryption key.

Declaration:
```
procedure DeKeyIdea(z: TIDEAKey;var 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 (420), CipherIdea (419)

17.4.3 EnKeyIdea

Synopsis: Create an IDEA encryption key from a user key.

Declaration: `procedure EnKeyIdea(UserKey: TIdeaCryptKey;var 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 (419), CipherIdea (419)

17.5 EIDEAError

17.5.1 Description

`EIDEAError` is used to signal errors in the IDEA encryption decryption streams.

17.6 TIDEADeCryptStream

17.6.1 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 (419) 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 (421) encryption stream.

17.6.2 Method overview

Page	Property	Description
420	Create	Constructor to create a new `TIDEADecryptStream` instance
421	Read	Reads data from the stream, decrypting it as needed
421	Seek	Set position on the stream

17.6.3 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 (419), which is then passed on to the inherited constructor TIDEAStream.Create (424). It is an easy-access function which introduces no new functionality.

The string is truncated at the maximum length of the TIdeaCryptKey (419) structure, so it makes no sense to provide a string with length longer than this structure.

See also: TIdeaCryptKey (419), TIDEAStream.Create (424), TIDEAEnCryptStream.Create (422)

17.6.4 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: TIDEADecryptStream.Write (420), TIDEADecryptStream.Seek (421), TIDEAEncryptStream (421)

17.6.5 TIDEADeCryptStream.Seek

Synopsis: Set position on the stream

Declaration: `function Seek(Offset: LongInt;Origin: Word) : LongInt;  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 (420) exception is raised if the stream does not allow the requested seek operation.

See also: TIDEADeCryptStream.Read (421)

17.7 TIDEAEncryptStream

17.7.1 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 (419) 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 (420) decryption stream.

17.7.2 Method overview

Page	Property	Description
422	Create	Constructor to create a new `TIDEAEncryptStream` instance
422	Destroy	Flush data buffers and free the stream instance.
423	Flush	Write remaining bytes from the stream
423	Seek	Set stream position
422	Write	Write bytes to the stream to be encrypted

17.7.3 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 (419), which is then passed on to the inherited constructor TIDEAStream.Create (424). It is an easy-access function which introduces no new functionality.

The string is truncated at the maximum length of the TIdeaCryptKey (419) structure, so it makes no sense to provide a string with length longer than this structure.

See also: TIdeaCryptKey (419), TIDEAStream.Create (424), TIDEADeCryptStream.Create (420)

17.7.4 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 (424)

17.7.5 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: TIDEADecryptStream.Read (421)

17.7.6 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 (420) will be raised unless called with `0` and `soFromCurrent` as arguments.

See also: TIDEAEncryptStream.Write (422), EIDEAError (420)

17.7.7 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: TIDEAEncryptStream.Write (422)

17.8 TIDEAStream

17.8.1 Description

Do not create instances of `TIDEAStream` directly. It implements no useful functionality: it serves as a common ancestor of the TIDEAEncryptStream (421) and TIDEADeCryptStream (420), 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.

17.8.2 Method overview

Page	Property	Description
424	Create	Creates a new instance of the `TIDEAStream` class

17.8.3 Property overview

Page	Property	Access	Description
424	Key	r	Key used when encrypting/decrypting

17.8.4 TIDEAStream.Create

Synopsis: Creates a new instance of the `TIDEAStream` class

Declaration: `constructor Create(AKey: TIDEAKey;Dest: TStream); Overload`

Visibility: public

Description: `Create` stores the encryption/decryption key and then calls the inherited `Create` to store the `Dest` stream.

Errors: None.

See also: TIDEAEncryptStream (421), TIDEADeCryptStream (420)

17.8.5 TIDEAStream.Key

Synopsis: Key used when encrypting/decrypting

Declaration: `Property Key : TIDEAKey`

Visibility: public

Access: Read

Description: `Key` is the key as it was passed to the constructor of the stream. It cannot be changed while data is read or written. It is the key as it is used when encrypting/decrypting.

See also: CipherIdea (419)

Chapter 18

Reference for unit 'inicol'

18.1 Used units

Table 18.1: Used units by unit 'inicol'

Name	Page
Classes	??
Inifiles	425
sysutils	??

18.2 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 (435) class for this.

18.3 Constants, types and variables

18.3.1 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.

18.4 EIniCol

18.4.1 Description

`EIniCol` is used to report error conditions in the load and save methods of TIniCollection (426).

18.5 TIniCollection

18.5.1 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 (429).

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.

18.5.2 Method overview

Page	Property	Description
426	Load	Loads the collection from the default filename.
428	LoadFromFile	Load collection from file.
428	LoadFromIni	Load collection from a file in .ini file format.
427	Save	Save the collection to the default filename.
427	SaveToFile	Save collection to a file in .ini file format
427	SaveToIni	Save the collection to a `TCustomIniFile` descendent

18.5.3 Property overview

Page	Property	Access	Description
429	FileName	rw	Filename of the collection
429	GlobalSection	rw	Name of the global section
428	Prefix	r	Prefix used in global section
429	SectionPrefix	r	Prefix string for section names

18.5.4 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 (429) property. It calls the LoadFromFile (428) 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 (428), TIniCollection.LoadFromIni (428), TIniCollection.Save (427), TIniCollection.FileName (429)

18.5.5 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 (429) property, using GlobalSection (429) as the section. It calls the SaveToFile (427) 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 (427), TIniCollection.SaveToIni (427), TIniCollection.Load (426), TIniCollection.FileName (429)

18.5.6 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 (428) property for the key. After this it calls the SaveToIni (430) method of all TIniCollectionItem (429) 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 (429) property contains the value of `Section`, it will be used in the Save (429) method.

See also: TIniCollectionItem.SaveToIni (430)

18.5.7 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 (427) 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 (427), TIniCollection.LoadFromFile (428)

18.5.8 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 (430) 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 (427) method description.

See also: TIniCollection.LoadFromFile (428), TIniCollectionItem.LoadFromIni (430), TIniCollection.SaveToIni (427)

18.5.9 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 (428) using this instance and `Section` as the parameters.

See also: TIniCollection.LoadFromIni (428), TIniCollection.Load (426), TIniCollection.SaveToIni (427), TIniCollection.SaveToFile (427)

18.5.10 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 (429), TIniCollection.GlobalSection (429)

18.5.11 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 (431) property to return the exact section name. It can be empty.

See also: TIniCollection.Section (426), TIniCollection.GlobalSection (429)

18.5.12 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 (428) or SaveToFile (427) operation. It is used in the Load (426) or Save (427) calls.

See also: TIniCollection.Save (427), TIniCollection.LoadFromFile (428), TIniCollection.SaveToFile (427), TIniCollection.Load (426)

18.5.13 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 (428) or SaveToIni (427) calls. It's used in the Load (426) or Save (427) calls.

See also: TIniCollection.Save (427), TIniCollection.LoadFromFile (428), TIniCollection.SaveToFile (427), TIniCollection.Load (426)

18.6 TIniCollectionItem

18.6.1 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 (430) and LoadFromIni (430) methods should be overridden. They should implement the actual loading and saving. The loading and saving is always initiated by the methods in TIniCollection (426), TIniCollection.LoadFromIni (428) and TIniCollection.SaveToIni (427) respectively.

18.6.2 Method overview

Page	Property	Description
431	LoadFromFile	Load item from a file
430	LoadFromIni	Method called when the item must be loaded
430	SaveToFile	Save item to a file
430	SaveToIni	Method called when the item must be saved

18.6.3 Property overview

Page	Property	Access	Description
431	SectionName	rw	Default section name

18.6.4 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 (427) 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 (427), TIniCollectionItem.LoadFromIni (430), TIniCollectionItem.SaveToFile (430), TIniCollectionItem.LoadFromFile (431)

18.6.5 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 (428) 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 (428), TIniCollectionItem.SaveToIni (430), TIniCollectionItem.LoadFromFile (431), TIniCollectionItem.SaveToFile (430)

18.6.6 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 (430) 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: TIniCollectionItem.SaveToIni (430), TIniCollectionItem.LoadFromFile (431)

18.6.7 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 (430) to load the item from the indicated file in .ini format from the section `Section`.

Errors: None.

See also: TIniCollectionItem.SaveToFile (430), TIniCollectionItem.LoadFromIni (430)

18.6.8 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: TIniCollectionItem.SaveToFile (430), TIniCollectionItem.LoadFromIni (430)

18.7 TNamedIniCollection

18.7.1 Description

`TNamedIniCollection` is the collection to go with the TNamedIniCollectionItem (433) item class. it provides some functions to look for items based on the UserData (432) or based on the Name (432).

18.7.2 Method overview

Page	Property	Description
432	FindByName	Return the item based on its name
433	FindByUserData	Return the item based on its `UserData`
432	IndexOfName	Search for an item, based on its name, and return its position
432	IndexOfUserData	Search for an item based on it's `UserData` property

18.7.3 Property overview

Page	Property	Access	Description
433	NamedItems	rw	Indexed access to the `TNamedIniCollectionItem` items

18.7.4 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 (432) 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: TNamedIniCollection.IndexOfName (432), TNamedIniCollectionItem.UserData (433)

18.7.5 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 insentitive). 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: TNamedIniCollection.IndexOfUserData (432), TNamedIniCollectionItem.Name (434)

18.7.6 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 (432) 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: TNamedIniCollection.IndexOfName (432), TNamedIniCollection.FindByUserData (433)

18.7.7 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 (433) property value matches the `UserData` parameter. If no item is found, `Nil` is returned.

Errors: If no item is found, `Nil` is returned.

18.7.8 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 (433) items. The index is zero based.

See also: TNamedIniCollectionItem (433)

18.8 TNamedIniCollectionItem

18.8.1 Description

`TNamedIniCollectionItem` is a TIniCollectionItem (429) descent with a published name property. The name is used as the section name when saving the item to the ini file.

18.8.2 Property overview

Page	Property	Access	Description
434	Name	rw	Name of the item
433	UserData	rw	User-defined data

18.8.3 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`.

18.8.4 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 (433)

Chapter 19

Reference for unit 'IniFiles'

19.1 Used units

Table 19.1: Used units by unit 'IniFiles'

Name	Page
Classes	??
contnrs	98
sysutils	??

19.2 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 (435) and is implemented in TIniFile (447) and TMemIniFile (455). The API presented by these components is Delphi compatible.

19.3 TCustomIniFile

19.3.1 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 (447) or TMemIniFile (455).

Since `TCustomIniFile` is an abstract class, it should never be created directly. Instead, one of the `TIniFile` or `TMemIniFile` classes should be created.

19.3.2 Method overview

Page	Property	Description
436	Create	Instantiate a new instance of `TCustomIniFile`.
443	DeleteKey	Delete a key from a section
437	Destroy	Remove the `TCustomIniFile` instance from memory
443	EraseSection	Clear a section
440	ReadBinaryStream	Read binary data
438	ReadBool	
439	ReadDate	Read a date value
439	ReadDateTime	Read a Date/Time value
440	ReadFloat	Read a floating point value
438	ReadInteger	Read an integer value from the file
442	ReadSection	Read the key names in a section
443	ReadSections	Read the list of sections
443	ReadSectionValues	Read names and values of a section
437	ReadString	Read a string valued key
440	ReadTime	Read a time value
437	SectionExists	Check if a section exists.
444	UpdateFile	Update the file on disk
444	ValueExists	Check if a value exists
442	WriteBinaryStream	Write binary data
439	WriteBool	Write boolean value
441	WriteDate	Write date value
441	WriteDateTime	Write date/time value
441	WriteFloat	Write a floating-point value
438	WriteInteger	Write an integer value
438	WriteString	Write a string value
442	WriteTime	Write time value

19.3.3 Property overview

Page	Property	Access	Description
445	CaseSensitive	rw	Are key and section names case sensitive
445	EscapeLineFeeds	r	Should linefeeds be escaped ?
444	FileName	r	Name of the .ini file
445	StripQuotes	rw	Should quotes be stripped from string values

19.3.4 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: TCustomIniFile.Destroy (437)

19.3.5 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 (435)

19.3.6 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 (444)

19.3.7 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: TCustomIniFile.WriteString (438), TCustomIniFile.ReadInteger (438), TCustomIniFile.ReadBool (438), TCustomIniFile.ReadDate (439), TCustomIniFile.ReadDateTime (439), TCustomIniFile.ReadTime (440), TCustomIniFile.ReadFloat (440), TCustomIniFile.ReadBinaryStream (440)

19.3.8 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: TCustomIniFile.ReadString (437), TCustomIniFile.WriteInteger (438), TCustomIniFile.WriteBool (439), TCustomIniFile.WriteDate (441), TCustomIniFile.WriteDateTime (441), TCustomIniFile.WriteTime (442), TCustomIniFile.WriteFloat (441), TCustomIniFile.WriteBinaryStream (442)

19.3.9 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: TCustomIniFile.WriteInteger (438), TCustomIniFile.ReadString (437), TCustomIniFile.ReadBool (438), TCustomIniFile.ReadDate (439), TCustomIniFile.ReadDateTime (439), TCustomIniFile.ReadTime (440), TCustomIniFile.ReadFloat (440), TCustomIniFile.ReadBinaryStream (440)

19.3.10 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: TCustomIniFile.ReadInteger (438), TCustomIniFile.WriteString (438), TCustomIniFile.WriteBool (439), TCustomIniFile.WriteDate (441), TCustomIniFile.WriteDateTime (441), TCustomIniFile.WriteTime (442), TCustomIniFile.WriteFloat (441), TCustomIniFile.WriteBinaryStream (442)

19.3.11 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.

Errors:

See also: TCustomIniFile.WriteBool (439), TCustomIniFile.ReadInteger (438), TCustomIniFile.ReadString (437), TCustomIniFile.ReadDate (439), TCustomIniFile.ReadDateTime (439), TCustomIniFile.ReadTime (440), TCustomIniFile.ReadFloat (440), TCustomIniFile.ReadBinaryStream (440)

19.3.12 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: TCustomIniFile.ReadBool (438), TCustomIniFile.WriteInteger (438), TCustomIniFile.WriteString (438), TCustomIniFile.WriteDate (441), TCustomIniFile.WriteDateTime (441), TCustomIniFile.WriteTime (442), TCustomIniFile.WriteFloat (441), TCustomIniFile.WriteBinaryStream (442)

19.3.13 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.

Errors:

See also: TCustomIniFile.WriteDate (441), TCustomIniFile.ReadInteger (438), TCustomIniFile.ReadBool (438), TCustomIniFile.ReadString (437), TCustomIniFile.ReadDateTime (439), TCustomIniFile.ReadTime (440), TCustomIniFile.ReadFloat (440), TCustomIniFile.ReadBinaryStream (440)

19.3.14 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: TCustomIniFile.WriteDateTime (441), TCustomIniFile.ReadInteger (438), TCustomIniFile.ReadBool (438), TCustomIniFile.ReadDate (439), TCustomIniFile.ReadString (437), TCustomIniFile.ReadTime (440), TCustomIniFile.ReadFloat (440), TCustomIniFile.ReadBinaryStream (440)

19.3.15 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: TCustomIniFile.WriteFloat (441), TCustomIniFile.ReadInteger (438), TCustomIniFile.ReadBool (438), TCustomIniFile.ReadDate (439), TCustomIniFile.ReadDateTime (439), TCustomIniFile.ReadTime (440), TCustomIniFile.ReadString (437), TCustomIniFile.ReadBinaryStream (440)

19.3.16 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.

Errors:

See also: TCustomIniFile.WriteTime (442), TCustomIniFile.ReadInteger (438), TCustomIniFile.ReadBool (438), TCustomIniFile.ReadDate (439), TCustomIniFile.ReadDateTime (439), TCustomIniFile.ReadString (437), TCustomIniFile.ReadFloat (440), TCustomIniFile.ReadBinaryStream (440)

19.3.17 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: TCustomIniFile.WriteBinaryStream (442), TCustomIniFile.ReadInteger (438), TCustomIniFile.ReadBool (438), TCustomIniFile.ReadDate (439), TCustomIniFile.ReadDateTime (439), TCustomIniFile.ReadTime (440), TCustomIniFile.ReadFloat (440), TCustomIniFile.ReadString (437)

19.3.18 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.

Errors:

See also: TCustomIniFile.ReadDate (439), TCustomIniFile.WriteInteger (438), TCustomIniFile.WriteBool (439), TCustomIniFile.WriteString (438), TCustomIniFile.WriteDateTime (441), TCustomIniFile.WriteTime (442), TCustomIniFile.WriteFloat (441), TCustomIniFile.WriteBinaryStream (442)

19.3.19 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: TCustomIniFile.ReadDateTime (439), TCustomIniFile.WriteInteger (438), TCustomIniFile.WriteBool (439), TCustomIniFile.WriteDate (441), TCustomIniFile.WriteString (438), TCustomIniFile.WriteTime (442), TCustomIniFile.WriteFloat (441), TCustomIniFile.WriteBinaryStream (442)

19.3.20 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: TCustomIniFile.ReadFloat (440), TCustomIniFile.WriteInteger (438), TCustomIniFile.WriteBool (439), TCustomIniFile.WriteDate (441), TCustomIniFile.WriteDateTime (441), TCustomIniFile.WriteTime (442), TCustomIniFile.WriteString (438), TCustomIniFile.WriteBinaryStream (442)

19.3.21 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: TCustomIniFile.ReadTime (440), TCustomIniFile.WriteInteger (438), TCustomIniFile.WriteBool (439), TCustomIniFile.WriteDate (441), TCustomIniFile.WriteDateTime (441), TCustomIniFile.WriteString (438), TCustomIniFile.WriteFloat (441), TCustomIniFile.WriteBinaryStream (442)

19.3.22 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 it's size can be determined. The data stream is not repositioned, it must be at the correct position.

See also: TCustomIniFile.ReadBinaryStream (440), TCustomIniFile.WriteInteger (438), TCustomIniFile.WriteBool (439), TCustomIniFile.WriteDate (441), TCustomIniFile.WriteDateTime (441), TCustomIniFile.WriteTime (442), TCustomIniFile.WriteFloat (441), TCustomIniFile.WriteString (438)

19.3.23 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 (443) method should be used.

See also: TCustomIniFile.ReadSections (443), TCustomIniFile.SectionExists (437), TCustomIniFile.ReadSectionValues (443)

19.3.24 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: TCustomIniFile.SectionExists (437), TCustomIniFile.ReadSectionValues (443), TCustomIniFile.ReadSection (442)

19.3.25 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 (442) can be used.

See also: TCustomIniFile.SectionExists (437), TCustomIniFile.ReadSections (443), TCustomIniFile.ReadSection (442)

19.3.26 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: TCustomIniFile.SectionExists (437), TCustomIniFile.ReadSections (443), TCustomIniFile.DeleteKey (443)

19.3.27 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: TCustomIniFile.EraseSection (443)

19.3.28 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 immediatly be committed to disk with a call to `UpdateFile`. Since the whole file is written to disk, this may have serious impact on performance.

See also: TIniFile.CacheUpdates (450)

19.3.29 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: TCustomIniFile.SectionExists (437)

19.3.30 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: TCustomIniFile.Create (436)

19.3.31 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 (436), as the value of this property must be specified when the `TCustomIniFile` instance is created.

By default, `EscapeLineFeeds` is `False`.

See also: TCustomIniFile.Create (436), TCustomIniFile.CaseSensitive (445)

19.3.32 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: TCustomIniFile.EscapeLineFeeds (445)

19.3.33 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).

19.4 THashedStringList

19.4.1 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 (446) and IndexOfName (446) functions make use of these hash values to quicklier locate a value.

19.4.2 Method overview

Page	Property	Description
446	Create	Instantiates a new instance of `THashedStringList`
446	Destroy	Clean up instance
446	IndexOf	Returns the index of a string in the list of strings
446	IndexOfName	Return the index of a name in the list of name=value pairs

19.4.3 THashedStringList.Create

Synopsis: Instantiates a new instance of `THashedStringList`

Declaration: `constructor Create`

Visibility: public

Description: `Create` calls the inherited `Create`, and then instantiates the hash tables.

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

See also: THashedStringList.Destroy (446)

19.4.4 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 (446)

19.4.5 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 TStringList.IndexOf (435) method and uses the hash values to look for the location of `S`.

See also: TStringList.IndexOf (435), THashedStringList.IndexOfName (446)

19.4.6 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 TStrings.IndexOfName (435) method and uses the hash values of the names to look for the location of `Name`.

See also: TStrings.IndexOfName (435), THashedStringList.IndexOf (446)

19.5 TIniFile

19.5.1 Description

`TIniFile` is an implementation of TCustomIniFile (435) which does the same as TMemIniFile (455), 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`

19.5.2 Method overview

Page	Property	Description
447	Create	Create a new instance of `TIniFile`
449	DeleteKey	Delete key
447	Destroy	Remove the `TIniFile` instance from memory
449	EraseSection	
448	ReadSection	Read the key names in a section
448	ReadSectionRaw	Read raw section
449	ReadSections	Read section names
449	ReadSectionValues	
448	ReadString	Read a string
450	UpdateFile	Update the file on disk
448	WriteString	Write string to file

19.5.3 Property overview

Page	Property	Access	Description
450	CacheUpdates	rw	Should changes be kept in memory
450	Stream	r	Stream from which ini file was read

19.5.4 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 (435), TMemIniFile (455)

19.5.5 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: TCustomIniFile.UpdateFile (444), TIniFile.CacheUpdates (450)

19.5.6 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 (437) abstract method by looking at the in-memory copy of the ini file and returning the string found there.

See also: TCustomIniFile.ReadString (437)

19.5.7 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 (438) abstract method by writing the string to the in-memory copy of the ini file. If CacheUpdates (450) 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.

19.5.8 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 (442)

See also: TCustomIniFile.ReadSection (442), TIniFile.ReadSectionRaw (448)

19.5.9 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 (448), TCustomIniFile.ReadSection (442)

19.5.10 TIniFile.ReadSections

Synopsis: Read section names

Declaration:
```
procedure ReadSections(Strings: TStrings);  Override
```

Visibility: public

Description: `ReadSections` is the implementation of TCustomIniFile.ReadSections (443). It operates on the in-memory copy of the inifile, and places all section names in `Strings`.

See also: TIniFile.ReadSection (448), TCustomIniFile.ReadSections (443), TIniFile.ReadSectionValues (449)

19.5.11 TIniFile.ReadSectionValues

Synopsis:

Declaration:
```
procedure ReadSectionValues(const Section: String;Strings: TStrings)
                                ;  Override
```

Visibility: public

Description: `ReadSectionValues` is the implementation of TCustomIniFile.ReadSectionValues (443). 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 (448), TCustomIniFile.ReadSectionValues (443), TIniFile.ReadSections (449)

19.5.12 TIniFile.EraseSection

Synopsis:

Declaration:
```
procedure EraseSection(const Section: String);  Override
```

Visibility: public

Description: `Erasesection` deletes the section `Section` from memory, if CacheUpdates (450) is `False`, then the file is immediatly updated on disk. This method is the implementation of the abstract TCustomIniFile.EraseSection (443) method.

See also: TCustomIniFile.EraseSection (443), TIniFile.ReadSection (448), TIniFile.ReadSections (449)

19.5.13 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 (450) is `False`, then the file is immediatly updated on disk.

See also: TIniFile.CacheUpdates (450)

19.5.14 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: TIniFile.CacheUpdates (450)

19.5.15 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 (450) method will use this stream to write changes to.

See also: TIniFile.Create (447), TIniFile.UpdateFile (450)

19.5.16 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 (450). 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 aversely affect performance.

See also: TIniFile.UpdateFile (450)

19.6 TIniFileKey

19.6.1 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 (447) class.

19.6.2 Method overview

Page	Property	Description
451	Create	Create a new instance of `TIniFileKey`

19.6.3 Property overview

Page	Property	Access	Description
451	Ident	rw	Key name
451	Value	rw	Key value

19.6.4 TIniFileKey.Create

Synopsis: Create a new instance of `TIniFileKey`

Declaration:
```
constructor Create(AIdent: String;AValue: String)
```

Visibility: public

Description: `Create` instantiates a new instance of `TIniFileKey` on the heap. It fills Ident (451) with `AIdent` and Value (451) with `AValue`.

See also: TIniFileKey.Ident (451), TIniFileKey.Value (451)

19.6.5 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: TIniFileKey.Value (451)

19.6.6 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: TIniFileKey.Ident (451)

19.7 TIniFileKeyList

19.7.1 Description

`TIniFileKeyList` maintains a list of TIniFileKey (451) instances on behalf of the TIniFileSection (453) class. It stores they keys of one section of the .ini files.

19.7.2 Method overview

Page	Property	Description
452	Clear	Clear the list
452	Destroy	Free the instance

19.7.3 Property overview

Page	Property	Access	Description
452	Items	r	Indexed access to `TIniFileKey` items in the list

19.7.4 TIniFileKeyList.Destroy

Synopsis: Free the instance

Declaration: `destructor Destroy;  Override`

Visibility: public

Description: `Destroy` clears up the list using Clear (452) and then calls the inherited destroy.

See also: TIniFileKeyList.Clear (452)

19.7.5 TIniFileKeyList.Clear

Synopsis: Clear the list

Declaration: `procedure Clear;  Override`

Visibility: public

Description: `Clear` removes all TIniFileKey (451) instances from the list, and frees the instances.

See also: TIniFileKey (451)

19.7.6 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 (451) items in the list. The index is zero-based and runs from 0 to `Count-1`.

See also: TIniFileKey (451)

19.8 TIniFileSection

19.8.1 Description

`TIniFileSection` is a class which represents a section in the .ini, and is used internally by the TIniFile (447) class (one instance of `TIniFileSection` is created for each section in the file by the TIniFileSectionList (454) list). The name of the section is stored in the Name (454) property, and the key/value pairs in this section are available in the KeyList (454) property.

19.8.2 Method overview

Page	Property	Description
453	Create	Create a new section object
453	Destroy	Free the section object from memory
453	Empty	Is the section empty

19.8.3 Property overview

Page	Property	Access	Description
454	KeyList	r	List of key/value pairs in this section
454	Name	r	Name of the section

19.8.4 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.

19.8.5 TIniFileSection.Create

Synopsis: Create a new section object

Declaration:
```
constructor Create(AName: String)
```

Visibility: public

Description: `Create` instantiates a new `TIniFileSection` class, and sets the name to `AName`. It allocates a TIniFileKeyList (452) instance to keep all the key/value pairs for this section.

See also: TIniFileKeyList (452)

19.8.6 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: TIniFileSection.Create (453), TIniFileKeyList (452)

19.8.7 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 (454)

19.8.8 TIniFileSection.KeyList

Synopsis: List of key/value pairs in this section

Declaration: `Property KeyList : TIniFileKeyList`

Visibility: public

Access: Read

Description: `KeyList` is the TIniFileKeyList (452) instance that is used by the `TIniFileSection` to keep the key/value pairs of the section.

See also: TIniFileSection.Name (454), TIniFileKeyList (452)

19.9 TIniFileSectionList

19.9.1 Description

`TIniFileSectionList` maintains a list of TIniFileSection (453) instances, one for each section in an .ini file. `TIniFileSectionList` is used internally by the TIniFile (447) class to represent the sections in the file.

19.9.2 Method overview

Page	Property	Description
455	Clear	Clear the list
455	Destroy	Free the object from memory

19.9.3 Property overview

Page	Property	Access	Description
455	Items	r	Indexed access to all the section objects in the list

19.9.4 TIniFileSectionList.Destroy

Synopsis: Free the object from memory

Declaration: `destructor Destroy;  Override`

Visibility: public

Description: `Destroy` calls Clear (455) to clear the section list and the calls the inherited `Destroy`

See also: TIniFileSectionList.Clear (455)

19.9.5 TIniFileSectionList.Clear

Synopsis: Clear the list

Declaration: `procedure Clear;  Override`

Visibility: public

Description: `Clear` removes all TIniFileSection (453) items from the list, and frees the items it removes from the list.

See also: TIniFileSection (453), TIniFileSectionList.Items (455)

19.9.6 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 (453), TIniFileSectionList.Clear (455)

19.10 TMemIniFile

19.10.1 Description

`TMemIniFile` is a simple descendent of TIniFile (447) 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.

19.10.2 Method overview

Page	Property	Description
456	Clear	Clear the data
456	Create	Create a new instance of `TMemIniFile`
456	GetStrings	Get contents of ini file as stringlist
456	Rename	Rename the ini file
457	SetStrings	Set data from a stringlist

19.10.3 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 (447), and sets the CacheUpdates (450) to `True` so updates will be kept in memory till they are explicitly written to disk.

See also: TIniFile.Create (447), TIniFile.CacheUpdates (450)

19.10.4 TMemIniFile.Clear

Synopsis: Clear the data

Declaration:
```
procedure Clear
```

Visibility: public

Description: `Clear` removes all sections and key/value pairs from memory. If CacheUpdates (450) is set to `False` then the file on disk will immediatly be emptied.

See also: TMemIniFile.SetStrings (457), TMemIniFile.GetStrings (456)

19.10.5 TMemIniFile.GetStrings

Synopsis: Get contents of ini file as stringlist

Declaration:
```
procedure GetStrings(List: TStrings)
```

Visibility: public

Description: `GetStrings` returns the whole contents of the ini file in a single stringlist, `List`. This includes comments and empty sections.

The `GetStrings` call can be used to get data for a call to SetStrings (457), which can be used to copy data between 2 in-memory ini files.

See also: TMemIniFile.SetStrings (457), TMemIniFile.Clear (456)

19.10.6 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: TIniFile.UpdateFile (450)

19.10.7 TMemIniFile.SetStrings

Synopsis: Set data from a stringlist

Declaration: `procedure SetStrings(List: TStrings)`

Visibility: public

Description: `SetStrings` sets the in-memory data from the `List` stringlist. The data is first cleared.

The `SetStrings` call can be used to set the data of the ini file to a list of strings obtained with GetStrings (456). The two calls combined can be used to copy data between 2 in-memory ini files.

See also: TMemIniFile.GetStrings (456), TMemIniFile.Clear (456)

Chapter 20

Reference for unit 'iostream'

20.1 Used units

Table 20.1: Used units by unit 'iostream'

Name	Page
Classes	**??**

20.2 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`).

20.3 Constants, types and variables

20.3.1 Types

```
TIOSType = (iosInput,iosOutPut,iosError)
```

Table 20.2: Enumeration values for type TIOSType

Value	Explanation
iosError	The stream can be used to write to standard diagnostic output
iosInput	The stream can be used to read from standard input
iosOutPut	The stream can be used to write to standard output

`TIOSType` is passed to the Create (459) constructor of TIOStream (459), it determines what kind of stream is created.

20.4 EIOStreamError

20.4.1 Description

Error thrown in case of an invalid operation on a TIOStream (459).

20.5 TIOStream

20.5.1 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 (458) 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 (460) behaviour based on this position.

20.5.2 Method overview

Page	Property	Description
459	Create	Construct a new instance of TIOStream (459)
459	Read	Read data from the stream.
460	Seek	Set the stream position
460	SetSize	Set the size of the stream
460	Write	Write data to the stream

20.5.3 TIOStream.Create

Synopsis: Construct a new instance of TIOStream (459)

Declaration: `constructor Create(aIOSType: TIOSType)`

Visibility: public

Description: `Create` creates a new instance of TIOStream (459), 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 (459), TIOStream.Write (460)

20.5.4 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 (459) 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 (458), TIOStream.Write (460)

20.5.5 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 (459) 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 (458), TIOStream.Read (459)

20.5.6 TIOStream.SetSize

Synopsis: Set the size of the stream

Declaration: `procedure SetSize(NewSize: LongInt);  Override`

Visibility: public

Description: `SetSize` overrides the standard `SetSize` implementation. It always raises an exception, because the standard input, output and stderr files have no size.

Errors: An `EIOStreamError` exception is raised when this method is called.

See also: EIOStreamError (459)

20.5.7 TIOStream.Seek

Synopsis: Set the stream position

Declaration: `function Seek(Offset: LongInt;Origin: Word) : LongInt;  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 (459) exception is raised if the stream does not allow the requested seek operation.

See also: EIOStreamError (459)

Chapter 21

Reference for unit 'libtar'

21.1 Used units

Table 21.1: Used units by unit 'libtar'

Name	Page
BaseUnix	461
Classes	??
sysutils	??
Unix	461
UnixType	461
Windows	461

21.2 Overview

The libtar units provides 2 classes to read and write .tar archives: TTarArchive (465) class can be used to read a tar file, and the TTarWriter (467) class can be used to write a tar file. The unit was implemented originally by Stefan Heymann.

21.3 Constants, types and variables

21.3.1 Constants

```
ALL_PERMISSIONS = [tpReadByOwner,tpWriteByOwner,tpExecuteByOwner,tpReadByGroup,tpWri
```

ALL_PERMISSIONS is a set constant containing all possible permissions (read/write/execute, for all groups of users) for an archive entry.

```
EXECUTE_PERMISSIONS = [tpExecuteByOwner,tpExecuteByGroup,tpExecuteByOther]
```

WRITE_PERMISSIONS is a set constant containing all possible execute permissions set for an archive entry.

```
FILETYPE_NAME : Array[TFileType] of String = ('Regular','Link','Symbolic Link','Char
```

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.

21.3.2 Types

```
TFileType = (ftNormal,ftLink,ftSymbolicLink,ftCharacter,ftBlock,
             ftDirectory,ftFifo,ftContiguous,ftDumpDir,ftMultiVolume,
             ftVolumeHeader)
```

Table 21.2: 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 (463) record.

```
TTarDirRec = record
  Name : String;
  Size : Int64;
  DateTime : TDateTime;
  Permissions : TTarPermissions;
  FileType : TFileType;
  LinkName : String;
  UID : Integer;
  GID : Integer;
  UserName : String;
  GroupName : String;
  ChecksumOK : Boolean;
```

```
  Mode : TTarModes;
  Magic : String;
  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 (466) call.

```
TTarMode = (tmSetUid,tmSetGid,tmSaveText)
```

Table 21.3: 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 (463) record.

```
TTarModes= Set of (tmSaveText,tmSetGid,tmSetUid)
```

TTarModes denotes the full set of permission bits for the file in the field Mode field of the TTarDirRec (463) record.

```
TTarPermission = (tpReadByOwner,tpWriteByOwner,tpExecuteByOwner,
                  tpReadByGroup,tpWriteByGroup,tpExecuteByGroup,
                  tpReadByOther,tpWriteByOther,tpExecuteByOther)
```

Table 21.4: 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 it stored in the .tar archive. Each of these enumerated constants correspond with one of the permission bits from a unix file permission.

```
TTarPermissions= Set of (tpExecuteByGroup,tpExecuteByOther,
                         tpExecuteByOwner,tpReadByGroup,tpReadByOther,
                         tpReadByOwner,tpWriteByGroup,tpWriteByOther,
                         tpWriteByOwner)
```

TTarPermissions describes the complete set of permissions that a file has. It is used in the Permissions field of the TTarDirRec (463) record.

21.4 Procedures and functions

21.4.1 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 (463)

21.4.2 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.

21.4.3 FileTimeGMT

Synopsis: Extract filetime

Declaration:
```
function FileTimeGMT(FileName: String) : TDateTime;  Overload
function FileTimeGMT(SearchRec: TSearchRec) : TDateTime;  Overload
```

Visibility: default

Description: FileTimeGMT returns the timestamp of a filename (FileName must exist) or a search rec (TSearchRec) to a GMT representation that can be used in a tar entry.

See also: TTarDirRec (463)

21.4.4 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 (464)

21.5 TTarArchive

21.5.1 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.

21.5.2 Method overview

Page	Property	Description
465	Create	Create a new instance of the archive
465	Destroy	Destroy `TTarArchive` instance
466	FindNext	Find next archive entry
466	GetFilePos	Return current archive position
466	ReadFile	Read a file from the archive
465	Reset	Reset archive
467	SetFilePos	Set position in archive

21.5.3 TTarArchive.Create

Synopsis: Create a new instance of the archive

Declaration:
```
constructor Create(Stream: TStream);  Overload
constructor Create(Filename: String;FileMode: Word);  Overload
```

Visibility: public

Description: `Create` can be used to create a new instance of `TTarArchive` using either a `Stream`TStream (**??**) descendent or using a name of a file to open: `FileName`. In case of the filename, an open mode can be specified.

Errors: In case a filename is specified and the file cannot be opened, an exception will occur.

See also: TTarArchive.FindNext (466)

21.5.4 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 (465)

21.5.5 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 (465)

21.5.6 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 succeful, 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 (466)

21.5.7 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 succesfully 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.

21.5.8 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 (467), TTarArchive.Reset (465)

21.5.9 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 (465), TTarArchive.GetFilePos (466)

21.6 TTarWriter

21.6.1 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.

21.6.2 Method overview

Page	Property	Description
469	AddDir	Add directory to archive
468	AddFile	Add a file to the archive
470	AddLink	Add hard link to archive
468	AddStream	Add stream contents to archive.
469	AddString	Add string as file data
469	AddSymbolicLink	Add a symbolic link to the archive
470	AddVolumeHeader	Add volume header entry
467	Create	Create a new archive
468	Destroy	Close archive and clean up `TTarWriter`
470	Finalize	Finalize the archive

21.6.3 Property overview

Page	Property	Access	Description
471	GID	rw	Archive entry group ID
471	GroupName	rw	Archive entry group name
472	Magic	rw	Archive entry Magic constant
472	Mode	rw	Archive entry mode
470	Permissions	rw	Archive entry permissions
471	UID	rw	Archive entry user ID
471	UserName	rw	Archive entry user name

21.6.4 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 (468)

21.6.5 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 (470)

21.6.6 TTarWriter.AddFile

Synopsis: Add a file to the archive

Declaration:
```
procedure AddFile(Filename: String;TarFilename: String)
```

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 (468), TTarWriter.AddString (469), TTarWriter.AddLink (470), TTarWriter.AddSymbolicLink (469), TTarWriter.AddDir (469), TTarWriter.AddVolumeHeader (470)

21.6.7 TTarWriter.AddStream

Synopsis: Add stream contents to archive.

Declaration:
```
procedure AddStream(Stream: TStream;TarFilename: String;
                    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 (468), TTarWriter.AddString (469), TTarWriter.AddLink (470), TTarWriter.AddSymbolicLink (469), TTarWriter.AddDir (469), TTarWriter.AddVolumeHeader (470)

21.6.8 TTarWriter.AddString

Synopsis: Add string as file data

Declaration:
```
procedure AddString(Contents: String;TarFilename: String;
                    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 (468), TTarWriter.AddStream (468), TTarWriter.AddLink (470), TTarWriter.AddSymbolicLink (469), TTarWriter.AddDir (469), TTarWriter.AddVolumeHeader (470)

21.6.9 TTarWriter.AddDir

Synopsis: Add directory to archive

Declaration:
```
procedure AddDir(Dirname: String;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 (468), TTarWriter.AddStream (468), TTarWriter.AddLink (470), TTarWriter.AddSymbolicLink (469), TTarWriter.AddString (469), TTarWriter.AddVolumeHeader (470)

21.6.10 TTarWriter.AddSymbolicLink

Synopsis: Add a symbolic link to the archive

Declaration:
```
procedure AddSymbolicLink(Filename: String;Linkname: String;
                          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.

Errors:

See also: TTarWriter.AddFile (468), TTarWriter.AddStream (468), TTarWriter.AddLink (470), TTarWriter.AddDir (469), TTarWriter.AddString (469), TTarWriter.AddVolumeHeader (470)

21.6.11 TTarWriter.AddLink

Synopsis: Add hard link to archive

Declaration: `procedure AddLink(Filename: String;Linkname: String;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.

Errors:

See also: TTarWriter.AddFile (468), TTarWriter.AddStream (468), TTarWriter.AddSymbolicLink (469), TTarWriter.AddDir (469), TTarWriter.AddString (469), TTarWriter.AddVolumeHeader (470)

21.6.12 TTarWriter.AddVolumeHeader

Synopsis: Add volume header entry

Declaration: `procedure AddVolumeHeader(VolumeId: String;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.

Errors:

See also: TTarWriter.AddFile (468), TTarWriter.AddStream (468), TTarWriter.AddSymbolicLink (469), TTarWriter.AddDir (469), TTarWriter.AddString (469), TTarWriter.AddLink (470)

21.6.13 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 (468)

21.6.14 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 (463)

21.6.15 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 (463)

21.6.16 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 (463)

21.6.17 TTarWriter.UserName

Synopsis: Archive entry user name

Declaration: `Property UserName : String`

Visibility: public

Access: Read,Write

Description: `UserName` is used for the `UserName` field of the archive entries.

See also: TTarDirRec (463)

21.6.18 TTarWriter.GroupName

Synopsis: Archive entry group name

Declaration: `Property GroupName : String`

Visibility: public

Access: Read,Write

Description: `GroupName` is used for the `GroupName` field of the archive entries.

See also: TTarDirRec (463)

21.6.19 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 (463)

21.6.20 TTarWriter.Magic

Synopsis: Archive entry Magic constant

Declaration: `Property Magic : String`

Visibility: public

Access: Read,Write

Description: `Magic` is used for the `Magic` field of the archive entries.

See also: TTarDirRec (463)

Chapter 22

Reference for unit 'Pipes'

22.1 Used units

Table 22.1: Used units by unit 'Pipes'

Name	Page
Classes	??
sysutils	??

22.2 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.

22.3 Constants, types and variables

22.3.1 Constants

```
ENoSeekMsg = 'Cannot seek on pipes'
```

Constant used in EPipeSeek (474) exception.

```
EPipeMsg = 'Failed to create pipe.'
```

Constant used in EPipeCreation (474) exception.

22.4 Procedures and functions

22.4.1 CreatePipeHandles

Synopsis: Function to create a set of pipe handles

Declaration: `function CreatePipeHandles(var Inhandle: THandle;var OutHandle: THandle) : 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 (474)

22.4.2 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 (473), and if that call is succesfull, a pair of streams is created: `InPipe` and `OutPipe`.

Errors: If no pipe handles could be created, an EPipeCreation (474) exception is raised.

See also: CreatePipeHandles (473), TInputPipeStream (474), TOutputPipeStream (476)

22.5 EPipeCreation

22.5.1 Description

Exception raised when an error occurred during the creation of a pipe pair.

22.6 EPipeError

22.6.1 Description

Exception raised when an invalid operation is performed on a pipe stream.

22.7 EPipeSeek

22.7.1 Description

Exception raised when an invalid seek operation is attempted on a pipe.

22.8 TInputPipeStream

22.8.1 Description

`TInputPipeStream` is created by the CreatePipeStreams (474) 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.

22.8.2 Method overview

Page	Property	Description
476	Read	Read data from the stream to a buffer.
475	Seek	Set the current position of the stream
475	Write	Write data to the stream.

22.8.3 Property overview

Page	Property	Access	Description
476	NumBytesAvailable	r	Number of bytes available for reading.

22.8.4 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 ENoWritePipe (473) exception is raised when this function is called.

See also: TInputPipeStream.Read (476), TInputPipeStream.Seek (475)

22.8.5 TInputPipeStream.Seek

Synopsis: Set the current position of the stream

Declaration: `function Seek(Offset: LongInt;Origin: Word) : LongInt;  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 (474) exception is raised if the stream does not allow the requested seek operation.

See also: EPipeSeek (474), #rtl.classes.tstream.seek (**??**)

22.8.6 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: TInputPipeStream.Write (475), TInputPipeStream.Seek (475)

22.8.7 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 (476)

22.9 TOutputPipeStream

22.9.1 Description

`TOutputPipeStream` is created by the CreatePipeStreams (474) call to represent the writing end of a pipe. It is a TStream (**??**) descendent which does not allow reading.

22.9.2 Method overview

Page	Property	Description
477	Read	Read data from the stream.
476	Seek	Sets the position in the stream

22.9.3 TOutputPipeStream.Seek

Synopsis: Sets the position in the stream

Declaration: `function Seek(Offset: LongInt;Origin: Word) : LongInt;  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 (474) exception is raised if this method is called.

22.9.4 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 ENoReadPipe (473) exception is raised when this function is called.

See also: TOutputPipeStream.Seek (476)

Chapter 23

Reference for unit 'pooledmm'

23.1 Used units

Table 23.1: Used units by unit 'pooledmm'

Name	Page
Classes	??

23.2 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.

23.3 Constants, types and variables

23.3.1 Types

```
PPooledMemManagerItem = ^TPooledMemManagerItem
```

`PPooledMemManagerItem` is a pointer type, pointing to a TPooledMemManagerItem (479) item, used in a linked list.

```
TEnumItemsMethod = procedure(Item: Pointer) of object
```

`TEnumItemsMethod` is a prototype for the callback used in the TNonFreePooledMemManager.EnumerateItems (480) call. The parameter `Item` will be set to each of the pointers in the item list of TNonFreePooledMemManager (479).

```
TPooledMemManagerItem = record
  Next : PPooledMemManagerItem;
end
```

TPooledMemManagerItem is used internally by the TPooledMemManager (481) class to maintain the free list block. It simply points to the next free block.

23.4 TNonFreePooledMemManager

23.4.1 Description

TNonFreePooledMemManager keeps a list of fixed-size memory blocks in memory. Each block has the same size, making it suitable for storing a lot of records of the same type. It does not free the items stored in it, except when the list is cleared as a whole.

It allocates memory for the blocks in a exponential way, i.e. each time a new block of memory must be allocated, it's size is the double of the last block. The first block will contain 8 items.

23.4.2 Method overview

Page	Property	Description
479	Clear	Clears the memory
479	Create	Creates a new instance of TNonFreePooledMemManager
480	Destroy	Removes the TNonFreePooledMemManager instance from memory
480	EnumerateItems	Enumerate all items in the list
480	NewItem	Return a pointer to a new memory block

23.4.3 Property overview

Page	Property	Access	Description
480	ItemSize	r	Size of an item in the list

23.4.4 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 (480) is valid after a call to Clear

See also: TNonFreePooledMemManager.NewItem (480)

23.4.5 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 (480)

23.4.6 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 (479), TNonFreePooledMemManager.Clear (479)

23.4.7 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 (480). 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 (479)

23.4.8 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)

23.4.9 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 (479)

23.5 TPooledMemManager

23.5.1 Description

`TPooledMemManager` is a class which maintains a linked list of blocks, represented by the TPooledMemManagerItem (479) record. It should not be used directly, but should be descended from and the descendent should implement the actual memory manager.

23.5.2 Method overview

Page	Property	Description
481	Clear	Clears the list
481	Create	Creates a new instance of the `TPooledMemManager` class
481	Destroy	Removes an instance of `TPooledMemManager` class from memory

23.5.3 Property overview

Page	Property	Access	Description
483	AllocatedCount	r	Total number of allocated items in the list
482	Count	r	Number of items in the list
483	FreeCount	r	Number of free items in the list
483	FreedCount	r	Total number of freed items in the list.
482	MaximumFreeCountRatio	rw	Maximum ratio of free items over total items
482	MinimumFreeCount	rw	Minimum count of free items in the list

23.5.4 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 (483)

23.5.5 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 (481)

23.5.6 TPooledMemManager.Destroy

Synopsis: Removes an instance of `TPooledMemManager` class from memory

Declaration: `destructor Destroy;  Override`

Visibility: public

Description: `Destroy` calls Clear (481) and then calls the inherited destroy.

`Destroy` should never be called directly. Instead `Free` should be used, or `FreeAndNil`

See also: TPooledMemManager.Create (481)

23.5.7 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 (482)

23.5.8 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 (482)

23.5.9 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 (483), TPooledMemManager.AllocatedCount (483), TPooledMemManager.FreedCount (483)

23.5.10 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 (482), TPooledMemManager.AllocatedCount (483), TPooledMemManager.FreedCount (483)

23.5.11 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 (482), TPooledMemManager.FreeCount (483), TPooledMemManager.FreedCount (483)

23.5.12 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 (482), TPooledMemManager.FreeCount (483), TPooledMemManager.AllocatedCount (483)

Chapter 24

Reference for unit 'process'

24.1 Used units

Table 24.1: Used units by unit 'process'

Name	Page
Classes	??
Pipes	473
sysutils	??

24.2 Overview

The Process unit contains the code for the TProcess (486) 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.

24.3 Constants, types and variables

24.3.1 Types

```
TProcessOption = (poRunSuspended,poWaitOnExit,poUsePipes,
                  poStderrToOutPut,poNoConsole,poNewConsole,
                  poDefaultErrorMode,poNewProcessGroup,poDebugProcess,
                  poDebugOnlyThisProcess)
```

When a new process is started using TProcess.Execute (488), these options control the way the process is started. Note that not all options are supported on all platforms.

```
TProcessOptions= Set of (poDebugOnlyThisProcess,poDebugProcess,
                         poDefaultErrorMode,poNewConsole,
```

Table 24.2: 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.

```
                                poNewProcessGroup,poNoConsole,poRunSuspended,
                                poStderrToOutPut,poUsePipes,poWaitOnExit)
```

Set of TProcessOption (484).

```
TProcessPriority = (ppHigh,ppIdle,ppNormal,ppRealTime)
```

Table 24.3: 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)
```

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)
```

These options are mainly for Win32, and determine what should be done with the application once it's started.

Table 24.4: 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

Table 24.5: Enumeration values for type TStartupOption

Value	Explanation
suoUseCountChars	Use the console character width as specified in TProcess (486).
suoUseFillAttribute	Use the console fill attribute as specified in TProcess (486).
suoUsePosition	Use the window sizes as specified in TProcess (486).
suoUseShowWindow	Use the Show Window options specified in TShowWindowOption (485)
suoUseSize	Use the window sizes as specified in TProcess (486)

```
TStartupOptions= Set of (suoUseCountChars,suoUseFillAttribute,
                         suoUsePosition,suoUseShowWindow,suoUseSize)
```

Set of TStartUpOption (485).

24.4 EProcess

24.4.1 Description

Exception raised when an error occurs in a TProcess routine.

24.5 TProcess

24.5.1 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 (494) property to the full pathname of the program that should be executed, and call Execute (488). To determine whether the process is still running (i.e. has not stopped executing), the Running (498) property can be checked.

More advanced techniques can be used with the Options (496) settings.

24.5.2 Method overview

Page	Property	Description
489	CloseInput	Close the input stream of the process
489	CloseOutput	Close the output stream of the process
489	CloseStderr	Close the error stream of the process
488	Create	Create a new instance of the `TProcess` class.
488	Destroy	Destroy this instance of `TProcess`
488	Execute	Execute the program with the given options
489	Resume	Resume execution of a suspended process
490	Suspend	Suspend a running process
490	Terminate	Terminate a running process
490	WaitOnExit	Wait for the program to stop executing.

24.5.3 Property overview

Page	Property	Access	Description
494	Active	rw	Start or stop the process.
494	ApplicationName	rw	Name of the application to start
494	CommandLine	rw	Command-line to execute
495	ConsoleTitle	rw	Title of the console window
495	CurrentDirectory	rw	Working directory of the process.
495	Desktop	rw	Desktop on which to start the process.
496	Environment	rw	Environment variables for the new process
493	ExitStatus	r	Exit status of the process.
501	FillAttribute	rw	Color attributes of the characters in the console window (Windows only)
491	Handle	r	Handle of the process
494	InheritHandles	rw	Should the created process inherit the open handles of the current process.
492	Input	r	Stream connected to standard input of the process.
496	Options	rw	Options to be used when starting the process.
493	Output	r	Stream connected to standard output of the process.
497	Priority	rw	Priority at which the process is running.
491	ProcessHandle	r	Alias for Handle (491)
492	ProcessID	r	ID of the process.
498	Running	r	Determines wheter the process is still running.
498	ShowWindow	rw	Determines how the process main window is shown (Windows only)
497	StartupOptions	rw	Additional (Windows) startup options
493	Stderr	r	Stream connected to standard diagnostic output of the process.
491	ThreadHandle	r	Main process thread handle
492	ThreadID	r	ID of the main process thread
499	WindowColumns	rw	Number of columns in console window (windows only)
499	WindowHeight	rw	Height of the process main window
499	WindowLeft	rw	X-coordinate of the initial window (Windows only)
491	WindowRect	rw	Positions for the main program window.
500	WindowRows	rw	Number of rows in console window (Windows only)
500	WindowTop	rw	Y-coordinate of the initial window (Windows only)
500	WindowWidth	rw	Height of the process main window (Windows only)

24.5.4 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.

24.5.5 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: TProcess.Create (488)

24.5.6 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 (494), applying as much as of the specified options as supported on the current platform.

If the `poWaitOnExit` option is specified in Options (496), then the call will only return when the program has finished executing (or if an error occured). If this option is not given, the call returns immediatly, but the WaitOnExit (490) call can be used to wait for it to close, or the Running (498) call can be used to check whether it is still running.

The TProcess.Terminate (490) call can be used to terminate the program if it is still running, or the Suspend (490) call can be used to temporarily stop the program's execution.

The ExitStatus (493) function can be used to check the program's exit status, after it has stopped executing.

Errors: On error a EProcess (486) exception is raised.

See also: TProcess.Running (498), TProcess.WaitOnExit (490), TProcess.Terminate (490), TProcess.Suspend (490), TProcess.Resume (489), TProcess.ExitStatus (493)

24.5.7 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: TProcess.Input (492), TProcess.StdErr (493), TProcess.Output (493), TProcess.CloseOutput (489), TProcess.CloseStdErr (489)

24.5.8 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: TProcess.Output (493), TProcess.Input (492), TProcess.StdErr (493), TProcess.CloseInput (489), TProcess.CloseStdErr (489)

24.5.9 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: TProcess.Output (493), TProcess.Input (492), TProcess.StdErr (493), TProcess.CloseInput (489), TProcess.CloseStdErr (489)

24.5.10 TProcess.Resume

Synopsis: Resume execution of a suspended process

Declaration: `function Resume : Integer; Virtual`

Visibility: public

Description: `Resume` should be used to let a suspended process resume it's execution. It should be called in particular when the `poRunSuspended` flag is set in Options (496).

Errors: None.

See also: TProcess.Suspend (490), TProcess.Options (496), TProcess.Execute (488), TProcess.Terminate (490)

24.5.11 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 (489) call.

`Suspend` is fundamentally different from TProcess.Terminate (490) which actually stops the process.

Errors: On error, a nonzero result is returned.

See also: TProcess.Options (496), TProcess.Resume (489), TProcess.Terminate (490), TProcess.Execute (488)

24.5.12 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 (493), TProcess.Suspend (490), TProcess.Execute (488), TProcess.WaitOnExit (490)

24.5.13 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 (493), TProcess.Terminate (490), TProcess.Running (498)

24.5.14 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

24.5.15 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 (488) has been called. It is not reset after the process stopped.

See also: TProcess.ThreadHandle (491), TProcess.ProcessID (492), TProcess.ThreadID (492)

24.5.16 TProcess.ProcessHandle

Synopsis: Alias for Handle (491)

Declaration: `Property ProcessHandle : THandle`

Visibility: public

Access: Read

Description: `ProcessHandle` equals Handle (491) and is provided for completeness only.

See also: TProcess.Handle (491), TProcess.ThreadHandle (491), TProcess.ProcessID (492), TProcess.ThreadID (492)

24.5.17 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 (488) has been called. It is not reset after the process stopped.

See also: TProcess.Handle (491), TProcess.ProcessID (492), TProcess.ThreadID (492)

24.5.18 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 (488) has been called. It is not reset after the process stopped.

See also: TProcess.Handle (491), TProcess.ThreadHandle (491), TProcess.ThreadID (492)

24.5.19 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 proces 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 (488) has been called. It is not reset after the process stopped.

See also: TProcess.ProcessID (492), TProcess.Handle (491), TProcess.ThreadHandle (491)

24.5.20 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 (496).

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 (493), TProcess.StdErr (493), TProcess.Options (496), TProcessOption (484)

24.5.21 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 (496).

The `Output` stream also contains any data written to standard diagnostic output (`stderr`) when the `poStdErrToOutPut` flag is used in Options (496).

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 (492), TProcess.StdErr (493), TProcess.Options (496), TProcessOption (484)

24.5.22 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 (496).

The `Output` stream equals the Output (493) when the `poStdErrToOutPut` flag is used in Options (496).

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 (492), TProcess.Output (493), TProcess.Options (496), TProcessOption (484)

24.5.23 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 (498), TProcess.Terminate (490)

24.5.24 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 (492), TProcess.Output (493), TProcess.StdErr (493)

24.5.25 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 (488), TProcess.Terminate (490)

24.5.26 TProcess.ApplicationName

Synopsis: Name of the application to start

Declaration: `Property ApplicationName : String`

Visibility: published

Access: Read,Write

Description: `ApplicationName` is an alias for TProcess.CommandLine (494). It's mostly foruse in the Windows `CreateProcess` call. If `CommandLine` is not set, then `ApplicationName` will be used instead.

Note that either `CommandLine` or `ApplicationName` must be set prior to calling `Execute`.

See also: TProcess.CommandLine (494)

24.5.27 TProcess.CommandLine

Synopsis: Command-line to execute

Declaration: `Property CommandLine : String`

Visibility: published

Access: Read,Write

Description: `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 (494)

24.5.28 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 (499), TProcess.WindowRows (500)

24.5.29 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 (496)

24.5.30 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 (492), TProcess.Output (493), TProcess.StdErr (493)

24.5.31 TProcess.Environment

Synopsis: Environment variables for the new process

Declaration: `Property Environment : TStrings`

Visibility: published

Access: Read,Write

Description: `Environment` contains the environment for the new process; it's a list of `Name=Value` pairs, one per line.

If it is empty, the environment of the current process is passed on to the new process.

See also: TProcess.Options (496)

24.5.32 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 (488) call is made.

Table 24.6:

Option	Meaning
`poRunSuspended`	Start the process in suspended state.
`poWaitOnExit`	Wait for the process to terminate before returning.
`poUsePipes`	Use pipes to redirect standard input and output.
`poStderrToOutPut`	Redirect standard error to the standard output stream.
`poNoConsole`	Do not allow access to the console window for the process (Win32 only)
`poNewConsole`	Start a new console window for the process (Win32 only)
`poDefaultErrorMode`	Use default error handling.
`poNewProcessGroup`	Start the process in a new process group (Win32 only)
`poDebugProcess`	Allow debugging of the process (Win32 only)
`poDebugOnlyThisProcess`	Do not follow processes started by this process (Win32 only)

See also: TProcessOption (484), TProcessOptions (485), TProcess.Priority (497), TProcess.StartUpOptions (497)

24.5.33 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 24.7:

Priority	Meaning
`ppHigh`	The process runs at higher than normal priority.
`ppIdle`	The process only runs when the system is idle (i.e. has nothing else to do)
`ppNormal`	The process runs at normal priority.
`ppRealTime`	The process runs at real-time priority.

Note that not all priorities can be set by any user. Usually, only users witha dministrative 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 24.8:

Priority	Nice value
`ppHigh`	20
`ppIdle`	20
`ppNormal`	0
`ppRealTime`	-20

See also: TProcessPriority (485)

24.5.34 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.

See also: TProcess.ShowWindow (498), TProcess.WindowHeight (499), TProcess.WindowWidth (500), TProcess.WindowLeft (499), TProcess.WindowTop (500), TProcess.WindowColumns (499), TProcess.WindowRows (500), TProcess.FillAttribute (501)

Table 24.9:

Priority	Meaning
suoUseShowWindow	Use the Show Window options specified in ShowWindow (498)
suoUseSize	Use the specified window sizes
suoUsePosition	Use the specified window sizes.
suoUseCountChars	Use the specified console character width.
suoUseFillAttribute	Use the console fill attribute specified in FillAttribute (501).

24.5.35 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 (490), TProcess.Active (494), TProcess.ExitStatus (493)

24.5.36 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 24.10:

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

24.5.37 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 (497)

See also: TProcess.WindowHeight (499), TProcess.WindowWidth (500), TProcess.WindowLeft (499), TProcess.WindowTop (500), TProcess.WindowRows (500), TProcess.FillAttribute (501), TProcess.StartupOptions (497)

24.5.38 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 (497)

See also: TProcess.WindowWidth (500), TProcess.WindowLeft (499), TProcess.WindowTop (500), TProcess.WindowColumns (499), TProcess.WindowRows (500), TProcess.FillAttribute (501), TProcess.StartupOptions (497)

24.5.39 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 (497)

See also: TProcess.WindowHeight (499), TProcess.WindowWidth (500), TProcess.WindowTop (500), TProcess.WindowColumns (499), TProcess.WindowRows (500), TProcess.FillAttribute (501), TProcess.StartupOptions (497)

24.5.40 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 (497)

See also: TProcess.WindowHeight (499), TProcess.WindowWidth (500), TProcess.WindowLeft (499), TProcess.WindowTop (500), TProcess.WindowColumns (499), TProcess.FillAttribute (501), TProcess.StartupOptions (497)

24.5.41 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 (497)

See also: TProcess.WindowHeight (499), TProcess.WindowWidth (500), TProcess.WindowLeft (499), TProcess.WindowColumns (499), TProcess.WindowRows (500), TProcess.FillAttribute (501), TProcess.StartupOptions (497)

24.5.42 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 (497)

See also: TProcess.WindowHeight (499), TProcess.WindowLeft (499), TProcess.WindowTop (500), TProcess.WindowColumns (499), TProcess.WindowRows (500), TProcess.FillAttribute (501), TProcess.StartupOptions (497)

24.5.43 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 (499), TProcess.WindowWidth (500), TProcess.WindowLeft (499), TProcess.WindowTop (500), TProcess.WindowColumns (499), TProcess.WindowRows (500), TProcess.StartupOptions (497)

Chapter 25

Reference for unit 'rttiutils'

25.1 Used units

Table 25.1: Used units by unit 'rttiutils'

Name	Page
Classes	??
StrUtils	502
sysutils	??
typinfo	??

25.2 Overview

The rttiutils unit is a unit providing simplified access to the RTTI information from published properties using the TPropInfoList (504) 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 (507) class can be used for this. The implementation is based on the apputils unit from `RXLib` by *AO ROSNO* and *Master-Bank*

25.3 Constants, types and variables

25.3.1 Constants

```
sPropNameDelimiter : String = '_'
```

Separator used when constructing section/key names

25.3.2 Types

```
TEraseSectEvent = procedure(const ASection: String) of object
```

TEraseSectEvent is used by TPropsStorage (507) to clear a storage section, in a .ini file like fashion: The call should remove all keys in the section ASection, and remove the section from storage.

```
TFindComponentEvent = function(const Name: String) : TComponent
```

TFindComponentEvent should return the component instance for the component with name path Name. The name path should be relative to the global list of loaded components.

```
TReadStrEvent = function(const ASection: String;const Item: String;
                         const Default: String) : String of object
```

TReadStrEvent is used by TPropsStorage (507) to read strings from a storage mechanism, in a .ini file like fashion: The call should read the string in ASection with key Item, and if it does not exist, Default should be returned.

```
TWriteStrEvent = procedure(const ASection: String;const Item: String;
                           const Value: String) of object
```

TWriteStrEvent is used by TPropsStorage (507) to write strings to a storage mechanism, in a .ini file like fashion: The call should write the string Value in ASection with key Item. The section and key should be created if they didn't exist yet.

25.3.3 Variables

```
FindGlobalComponentCallBack : TFindComponentEvent
```

FindGlobalComponentCallBack is called by UpdateStoredList (504) 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.

25.4 Procedures and functions

25.4.1 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 (504), TPropsStorage.StoreObjectsProps (509) or TPropsStorage.LoadObjectsProps (508).

See also: ParseStoredItem (504), UpdateStoredList (504), TPropsStorage.StoreObjectsProps (509), TPropsStorage.LoadObjectsProps (508)

25.4.2 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 (503). 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 (503), UpdateStoredList (504), TPropsStorage.StoreObjectsProps (509), TPropsStorage.LoadObjectsProps (508)

25.4.3 UpdateStoredList

Synopsis: Update a stringlist with object references

Declaration:
```
procedure UpdateStoredList(AComponent: TComponent;AStoredList: TStrings;
                           FromForm: Boolean)
```

Visibility: default

Description: `UpdateStoredList` will parse the strings in `AStoredList` using ParseStoredItem (504) 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 (503) 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 (504), TPropsStorage.StoreObjectsProps (509), TPropsStorage.LoadObjectsProps (508), FindGlobalComponentCallBack (503)

25.5 TPropInfoList

25.5.1 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 (507)

25.5.2 Method overview

Page	Property	Description
505	Contains	Check whether a certain property is included
505	Create	Create a new instance of `TPropInfoList`
506	Delete	Delete property information from the list
505	Destroy	Remove the `TPropInfoList` instance from memory
505	Find	Retrieve property information based on name
506	Intersect	Intersect 2 property lists

25.5.3 Property overview

Page	Property	Access	Description
506	Count	r	Number of items in the list
506	Items	r	Indexed access to the property type pointers

25.5.4 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: TPropInfoList.Destroy (505)

25.5.5 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: TPropInfoList.Create (505)

25.5.6 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: TPropInfoList.Find (505), TPropInfoList.Intersect (506)

25.5.7 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: TPropInfoList.Intersect (506), TPropInfoList.Contains (505)

25.5.8 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 (506) call.

Errors: No checking on the validity of `Index` is performed.

See also: TPropInfoList.Intersect (506)

25.5.9 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: TPropInfoList.Delete (506), TPropInfoList.Contains (505)

25.5.10 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: TPropInfoList.Items (506)

25.5.11 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: TPropInfoList.Count (506)

25.6 TPropsStorage

25.6.1 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.

25.6.2 Method overview

Page	Property	Description
507	LoadAnyProperty	Load a property value
508	LoadObjectsProps	Load a list of component properties
508	LoadProperties	Load a list of properties
507	StoreAnyProperty	Store a property value
509	StoreObjectsProps	Store a list of component properties
508	StoreProperties	Store a list of properties

25.6.3 Property overview

Page	Property	Access	Description
510	AObject	rw	Object to load or store properties from
511	OnEraseSection	rw	Erase a section in storage
510	OnReadString	rw	Read a string value from storage
511	OnWriteString	rw	Write a string value to storage
510	Prefix	rw	Prefix to use in storage
510	Section	rw	Section name for storage

25.6.4 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 (510) property of `TPropsStorage`.

Errors: If the property pointer is invalid or `AObject` is invalid, an exception will be raised.

See also: TPropsStorage.AObject (510), TPropsStorage.LoadAnyProperty (507), TPropsStorage.LoadProperties (508), TPropsStorage.StoreProperties (508)

25.6.5 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 (510) property of `TPropsStorage`.

Errors: If the property pointer is invalid or `AObject` is invalid, an exception will be raised.

See also: TPropsStorage.AObject (510), TPropsStorage.StoreAnyProperty (507), TPropsStorage.LoadProperties (508), TPropsStorage.StoreProperties (508)

25.6.6 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 (510) object.

Errors: If an invalid property name is specified, an exception will be raised.

See also: TPropsStorage.AObject (510), TPropsStorage.StoreAnyProperty (507), TPropsStorage.LoadProperties (508), TPropsStorage.LoadAnyProperty (507)

25.6.7 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 (510) object.

Errors: If an invalid property name is specified, an exception will be raised.

See also: TPropsStorage.AObject (510), TPropsStorage.StoreAnyProperty (507), TPropsStorage.StoreProperties (508), TPropsStorage.LoadAnyProperty (507)

25.6.8 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 (504) call.

For example, to load the checked state of a checkbox named 'CBCheckMe' and the caption of a button named 'BPressMe', both owned by a form, the following strings should be passed:

```
CBCheckMe.Checked
BPressMe.Caption
```

and the `ACompontent` should be the form component that owns the button and checkbox.

Note that this call removes the value of the AObject (510) property.

Errors: If an invalid component is specified, an exception will be raised.

See also: UpdateStoredList (504), TPropsStorage.StoreObjectsProps (509), TPropsStorage.LoadProperties (508), TPropsStorage.LoadAnyProperty (507)

25.6.9 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 (504) call.

For example, to store the checked state of a checkbox named 'CBCheckMe' and the caption of a button named 'BPressMe', both owned by a form, the following strings should be passed:

```
CBCheckMe.Checked
BPressMe.Caption
```

and the `ACompontent` should be the form component that owns the button and checkbox.

Note that this call removes the value of the AObject (510) property.

See also: UpdateStoredList (504), TPropsStorage.LoadObjectsProps (508), TPropsStorage.LoadProperties (508), TPropsStorage.LoadAnyProperty (507)

25.6.10 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 (509) or LoadObjectProps (508) will destroy any value that this property might have.

See also: TPropsStorage.LoadProperties (508), TPropsStorage.LoadAnyProperty (507), TPropsStorage.StoreProperties (508), TPropsStorage.StoreAnyProperty (507), TPropsStorage.StoreObjectsProps (509), TPropsStorage.LoadObjectsProps (508)

25.6.11 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 (510)

25.6.12 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 (510)

25.6.13 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: TPropsStorage.OnWriteString (511), TPropsStorage.OnEraseSection (511), TReadStrEvent (503)

25.6.14 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: TPropsStorage.OnReadString (510), TPropsStorage.OnEraseSection (511), TWriteStrEvent (503)

25.6.15 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: TPropsStorage.OnReadString (510), TPropsStorage.OnWriteString (511), TEraseSectEvent (502)

Chapter 26

Reference for unit 'simpleipc'

26.1 Used units

Table 26.1: Used units by unit 'simpleipc'

Name	Page
Classes	??
sysutils	??

26.2 Overview

The SimpleIPC unit provides classes to implement a simple, one-way IPC mechanism using string messages. It provides a TSimpleIPCServer (522) component for the server, and a TSimpleIPCClient (519) component for the client. The components are cross-platform, and should work both on Windows and unix-like systems.

26.3 Constants, types and variables

26.3.1 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

26.3.2 Constants

```
MsgVersion = 1
```

Current version of the messaging protocol

```
mtString = 1
```

String message type

```
mtUnknown = 0
```

Unknown message type

26.3.3 Types

```
TIPCClientCommClass = Class of TIPCClientComm
```

`TIPCClientCommClass` is used by TSimpleIPCClient (519) to decide which kind of communication channel to set up.

```
TIPCServerCommClass = Class of TIPCServerComm
```

`TIPCServerCommClass` is used by TSimpleIPCServer (522) 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.

26.3.4 Variables

```
DefaultIPCClientClass : TIPCClientCommClass = nil
```

`DefaultIPCClientClass` is filled with a class pointer indicating which kind of communication protocol class should be instantiated by the TSimpleIPCClient (519) 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 (522) 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.

26.4 EIPCError

26.4.1 Description

`EIPCError` is the exception used by the various classes in the SimpleIPC unit to report errors.

26.5 TIPCClientComm

26.5.1 Description

`TIPCClientComm` is an abstract component which implements the client-side communication protocol. The behaviour expected of this class must be implemented in a platform-dependent descendent class.

The TSimpleIPCClient (519) class does not implement the messaging protocol by itself. Instead, it creates an instance of a (platform dependent) descendent of `TIPCClientComm` which handles the internals of the commnication protocol.

The server side of the messaging protocol is handled by the TIPCServerComm (516) component. The descenent components must always be implemented in pairs.

26.5.2 Method overview

Page	Property	Description
514	Connect	Connect to the server
514	Create	Create a new instance of the `TIPCClientComm`
515	Disconnect	Disconnect from the server
515	SendMessage	Send a message
515	ServerRunning	Check if the server is running.

26.5.3 Property overview

Page	Property	Access	Description
516	Owner	r	`TSimpleIPCClient` instance for which communication must be handled.

26.5.4 TIPCClientComm.Create

Synopsis: Create a new instance of the `TIPCClientComm`

Declaration: `constructor Create(AOwner: TSimpleIPCClient); Virtual`

Visibility: public

Description: `Create` instantiates a new instance of the `TIPCClientComm` class, and stores the `AOwner` reference to the TSimpleIPCClient (519) instance for which it will handle communitation. It can be retrieved later using the Owner (516) property.

See also: TIPCClientComm.Owner (516), TSimpleIPCClient (519)

26.5.5 TIPCClientComm.Connect

Synopsis: Connect to the server

Declaration: `procedure Connect; Virtual; Abstract`

Visibility: public

Description: `Connect` must establish a communication channel with the server. The server endpoint must be constructed from the ServerID (519) and ServerInstance (522) properties of the owning TSimpleIPCClient (519) instance.

`Connect` is called by the TSimpleIPCClient.Connect (520) call or when the Active (519) property is set to `True`

Messages can be sent only after `Connect` was called succesfully.

Errors: If the connection setup fails, or the connection was already set up , then an exception may be raised.

See also: TSimpleIPCClient.Connect (520), TSimpleIPC.Active (519), TIPCClientComm.Disconnect (515)

26.5.6 TIPCClientComm.Disconnect

Synopsis: Disconnect from the server

Declaration: `procedure Disconnect; Virtual; Abstract`

Visibility: public

Description: `Disconnect` closes the communication channel with the server. Any calls to `SendMessage` are invalid after `Disconnect` was called.

`Disconnect` is called by the TSimpleIPCClient.Disconnect (521) call or when the Active (519) property is set to `False`.

Messages can no longer be sent after `Disconnect` was called.

Errors: If the connection shutdown fails, or the connection was already shut down, then an exception may be raised.

See also: TSimpleIPCClient.Disconnect (521), TSimpleIPC.Active (519), TIPCClientComm.Connect (514)

26.5.7 TIPCClientComm.ServerRunning

Synopsis: Check if the server is running.

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

Visibility: public

Description: `ServerRunning` returns `True` if the server endpoint 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 TSimpleIPCClient (519) component.

See also: TSimpleIPCClient.InstanceID (519), TSimpleIPCClient.ServerID (519)

26.5.8 TIPCClientComm.SendMessage

Synopsis: Send a message

Declaration: `procedure SendMessage(MsgType: TMessageType;Stream: TStream); Virtual ; Abstract`

Visibility: public

Description: `SendMessage` should deliver the message with type `MsgType` and data in `Stream` to the server. It should not return until the message was delivered.

Errors: If the delivery of the message fails, an exception will be raised.

26.5.9 TIPCClientComm.Owner

Synopsis: `TSimpleIPCClient` instance for which communication must be handled.

Declaration: `Property Owner : TSimpleIPCClient`

Visibility: public

Access: Read

Description: `Owner` is the TSimpleIPCClient (519) instance for which the communication must be handled. It cannot be changed, and must be specified when the `TIPCClientComm` instance is created.

See also: TSimpleIPCClient (519), TIPCClientComm.Create (514)

26.6 TIPCServerComm

26.6.1 Description

`TIPCServerComm` is an abstract component which implements the server-side communication protocol. The behaviour expected of this class must be implemented in a platform-dependent descendent class.

The TSimpleIPCServer (522) class does not implement the messaging protocol by itself. Instead, it creates an instance of a (platform dependent) descendent of `TIPCServerComm` which handles the internals of the commnication protocol.

The client side of the messaging protocol is handled by the TIPCClientComm (514) component. The descenent components must always be implemented in pairs.

26.6.2 Method overview

Page	Property	Description
516	Create	Create a new instance of the communication handler
517	PeekMessage	See if a message is available.
518	ReadMessage	Read message from the channel.
517	StartServer	Start the server-side of the communication channel
517	StopServer	Stop the server side of the communication channel.

26.6.3 Property overview

Page	Property	Access	Description
518	InstanceID	r	Unique identifier for the communication channel.
518	Owner	r	`TSimpleIPCServer` instance for which to handle transport

26.6.4 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 (518) property.

See also: TIPCServerComm.Owner (518)

26.6.5 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 (519) property of the TSimpleIPCServer (522) instance is set to `True`.

Errors: In case of an error, an EIPCError (514) exception is raised.

See also: TSimpleIPCServer (522), TSimpleIPC.Active (519)

26.6.6 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 (519) property of the TSimpleIPCServer (522) instance is set to `False`.

Errors: In case of an error, an EIPCError (514) exception is raised.

See also: TSimpleIPCServer (522), TSimpleIPC.Active (519)

26.6.7 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 (518) call.

See also: TIPCServerComm.ReadMessage (518)

26.6.8 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 (517) to see if a message is available.

See also: TSimpleIPCServer (522)

26.6.9 TIPCServerComm.Owner

Synopsis: `TSimpleIPCServer` instance for which to handle transport

Declaration: `Property Owner : TSimpleIPCServer`

Visibility: public

Access: Read

Description: `Owner` refers to the TSimpleIPCServer (522) instance for which this instance of `TSimpleIPCServer` handles the transport. It is specified when the `TIPCServerComm` is created.

See also: TSimpleIPCServer (522)

26.6.10 TIPCServerComm.InstanceID

Synopsis: Unique identifier for the communication channel.

Declaration: `Property InstanceID : String`

Visibility: public

Access: Read

Description: `InstanceID` returns a textual representation which uniquely identifies the communication channel on the server. The value is system dependent, and should be usable by the client-side to establish a communication channel with this instance.

26.7 TSimpleIPC

26.7.1 Description

`TSimpleIPC` is the common ancestor for the TSimpleIPCServer (522) and TSimpleIPCClient (519) classes. It implements some common properties between client and server.

26.7.2 Property overview

Page	Property	Access	Description
519	Active	rw	Communication channel active
519	ServerID	rw	Unique server identification

26.7.3 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 (519) identification.

See also: TSimpleIPC.ServerID (519)

26.7.4 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: TSimpleIPC.Active (519)

26.8 TSimpleIPCClient

26.8.1 Description

`TSimpleIPCClient` is the client side of the simple IPC communication protocol. The client program should create a `TSimpleIPCClient` instance, set its ServerID (519) property to the unique name for the server it wants to send messages to, and then set the Active (519) property to `True` (or call Connect (519)).

After the connection with the server was established, messages can be sent to the server with the SendMessage (521) or SendStringMessage (521) calls.

26.8.2 Method overview

Page	Property	Description
520	Connect	Connect to the server
520	Create	Create a new instance of `TSimpleIPCClient`
520	Destroy	Remove the `TSimpleIPCClient` instance from memory
521	Disconnect	Disconnect from the server
521	SendMessage	Send a message to the server
521	SendStringMessage	Send a string message to the server
522	SendStringMessageFmt	Send a formatted string message
521	ServerRunning	Check if the server is running.

26.8.3 Property overview

Page	Property	Access	Description
522	ServerInstance	rw	Server instance identification

26.8.4 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: TSimpleIPCClient.Destroy (520)

26.8.5 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: TSimpleIPCClient.Create (520)

26.8.6 TSimpleIPCClient.Connect

Synopsis: Connect to the server

Declaration: `procedure Connect`

Visibility: public

Description: `Connect` connects to the server indicated in the ServerID (519) and InstanceID (519) properties. `Connect` is called automatically if the Active (519) property is set to `True`.

After a successful call to `Connect`, messages can be sent to the server using SendMessage (521) or SendStringMessage (521).

Calling `Connect` if the connection is already open has no effect.

Errors: If creating the connection fails, an EIPCError (514) exception may be raised.

See also: TSimpleIPC.ServerID (519), TSimpleIPCClient.InstanceID (519), TSimpleIPC.Active (519), TSimpleIPCClient.SendMessage (521), TSimpleIPCClient.SendStringMessage (521), TSimpleIPCClient.Disconnect (521)

26.8.7 TSimpleIPCClient.Disconnect

Synopsis: Disconnect from the server

Declaration: `procedure Disconnect`

Visibility: public

Description: `Disconnect` shuts down the connection with the server as previously set up with Connect (520). `Disconnect` is called automatically if the Active (519) property is set to `False`.

After a successful call to `Disconnect`, messages can no longer be sent to the server. Attempting to do so will result in an exception.

Calling `Disconnect` if there is no connection has no effect.

Errors: If creating the connection fails, an EIPCError (514) exception may be raised.

See also: TSimpleIPC.Active (519), TSimpleIPCClient.Connect (520)

26.8.8 TSimpleIPCClient.ServerRunning

Synopsis: Check if the server is running.

Declaration: `function ServerRunning : Boolean`

Visibility: public

Description: `ServerRunning` verifies if the server indicated in the ServerID (519) and InstanceID (519) 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: TSimpleIPCClient.Connect (520)

26.8.9 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 EIPCError (514) exception is raised.

See also: TSimpleIPCClient.Connect (520), TSimpleIPCClient.SendStringMessage (521)

26.8.10 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 `MsgTyp` and data `Msg` to the server. This is a convenience function: a small wrapper around the SendMessage (521) method

Errors: Same as for `SendMessage`.

See also: TSimpleIPCClient.SendMessage (521), TSimpleIPCClient.Connect (520), TSimpleIPCClient.SendStringMessageFmt (522)

26.8.11 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 `MsgTyp` and message formatted from `Msg` and `Args` to the server. This is a convenience function: a small wrapper around the SendStringMessage (521) method

Errors: Same as for `SendMessage`.

See also: TSimpleIPCClient.SendMessage (521), TSimpleIPCClient.Connect (520), TSimpleIPCClient.SendStringMessage (521)

26.8.12 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 (526) property set to `False`, and should match the server's InstanceID (525) property.

See also: TSimpleIPC.ServerID (519), TSimpleIPCServer.Global (526), TSimpleIPCServer.InstanceID (525)

26.9 TSimpleIPCServer

26.9.1 Description

`TSimpleIPCServer` is the server side of the simple IPC communication protocol. The server program should create a `TSimpleIPCServer` instance, set its ServerID (519) property to a unique name for the system, and then set the Active (519) property to `True` (or call StartServer (523)).

After the server was started, it can check for availability of messages with the PeekMessage (524) call, and read the message with ReadMessage (522).

26.9.2 Method overview

Page	Property	Description
523	Create	Create a new instance of `TSimpleIPCServer`
523	Destroy	Remove the `TSimpleIPCServer` instance from memory
524	GetMessageData	Read the data of the last message in a stream
524	PeekMessage	Check if a client message is available.
523	StartServer	Start the server
524	StopServer	Stop the server

26.9.3 Property overview

Page	Property	Access	Description
526	Global	rw	Is the server reachable to all users or not
525	InstanceID	r	Instance ID
525	MsgData	r	Last message data
525	MsgType	r	Last message type
526	OnMessage	rw	Event triggered when a pessage arrives
525	StringMessage	r	Last message as a string.

26.9.4 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: TSimpleIPCServer.Destroy (523)

26.9.5 TSimpleIPCServer.Destroy

Synopsis: Remove the `TSimpleIPCServer` instance from memory

Declaration: `destructor Destroy; Override`

Visibility: public

Description: `Destroy` stops the server, cleans up the internal data structures maintained by `TSimpleIPCServer` and then calls the inherited `Destroy`, which will remove the instance from memory.

Never call `Destroy` directly, use the `Free` method instead or the `FreeAndNil` procedure in SysUtils.

See also: TSimpleIPCServer.Create (523)

26.9.6 TSimpleIPCServer.StartServer

Synopsis: Start the server

Declaration: `procedure StartServer`

Visibility: public

Description: `StartServer` starts the server side of the communication channel. It is called automatically when the `Active` property is set to `True`. It creates the internal communication object (a TIPCServerComm (516) descendent) and activates the communication channel.

After this method was called, clients can connect and send messages.

Prior to calling this method, the ServerID (519) property must be set.

Errors: If an error occurs a EIPCError (514) exception may be raised.

See also: TIPCServerComm (516), TSimpleIPC.Active (519), TSimpleIPC.ServerID (519), TSimpleIPCServer.StopServer (524)

26.9.7 TSimpleIPCServer.StopServer

Synopsis: Stop the server

Declaration: `procedure StopServer`

Visibility: public

Description: `StopServer` stops the server side of the communication channel. It is called automatically when the `Active` property is set to `False`. It deactivates the communication channel and frees the internal communication object (a TIPCServerComm (516) descendent).

See also: TIPCServerComm (516), TSimpleIPC.Active (519), TSimpleIPC.ServerID (519), TSimpleIPCServer.StartServer (523)

26.9.8 TSimpleIPCServer.PeekMessage

Synopsis: Check if a client message is available.

Declaration: `function PeekMessage(TimeOut: Integer;DoReadMessage: Boolean) : Boolean`

Visibility: public

Description: `PeekMessage` checks if a message from a client is available. It will return `True` if a message is available. The call will wait for `TimeOut` milliseconds for a message to arrive: if after `TimeOut` milliseconds, no message is available, the function will return `False`.

If `DoReadMessage` is `True` then `PeekMessage` will read the message. If it is `False`, it does not read the message. The message should then be read manually with ReadMessage (522).

See also: TSimpleIPCServer.ReadMessage (522)

26.9.9 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 (525) 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 (522). It will also not clear the data buffer.

See also: TSimpleIPCServer.StringMessage (525), TSimpleIPCServer.MsgData (525), TSimpleIPCServer.MsgType (525)

26.9.10 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 (522).

See also: TSimpleIPCServer.GetMessageData (524)

26.9.11 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 (522).

See also: TSimpleIPCServer.ReadMessage (522)

26.9.12 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 (525) is better suited to read the data.

This property will contain valid data only after a succesful call to ReadMessage (522).

See also: TSimpleIPCServer.StringMessage (525), TSimpleIPCServer.ReadMessage (522)

26.9.13 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 (522) property to form the unique server endpoint which a client should use.

See also: TSimpleIPCServer.ServerID (522), TSimpleIPCServer.GlobalID (522)

26.9.14 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: TSimpleIPCServer.ServerID (522), TSimpleIPCServer.InstanceID (525)

26.9.15 TSimpleIPCServer.OnMessage

Synopsis: Event triggered when a pessage arrives

Declaration: `Property OnMessage : TNotifyEvent`

Visibility: published

Access: Read,Write

Description: `OnMessage` is called by ReadMessage (522) when a message has been read. The actual message data can be retrieved with one of the StringMessage (525), MsgData (525) or MsgType (525) properties.

See also: TSimpleIPCServer.StringMessage (525), TSimpleIPCServer.MsgData (525), TSimpleIPCServer.MsgType (525)

Chapter 27

Reference for unit 'streamcoll'

27.1 Used units

Table 27.1: Used units by unit 'streamcoll'

Name	Page
Classes	??
sysutils	??

27.2 Overview

The streamcoll unit contains the implentation 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 (530) 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.

27.3 Procedures and functions

27.3.1 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 (529) and returns the read value. The value cannot be read and written across systems that have different endian values.

See also: ColReadDateTime (528), ColWriteBoolean (529), ColReadString (529), ColReadInteger (528), ColReadFloat (528), ColReadCurrency (528)

27.3.2 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 (529) and returns the read value. The value cannot be read and written across systems that have different endian values.

See also: ColReadDateTime (528), ColReadBoolean (527), ColReadString (529), ColReadInteger (528), ColReadFloat (528), ColWriteCurrency (529)

27.3.3 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 (529) and returns the read value. The value cannot be read and written across systems that have different endian values.

See also: ColWriteDateTime (529), ColReadBoolean (527), ColReadString (529), ColReadInteger (528), ColReadFloat (528), ColReadCurrency (528)

27.3.4 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 (530) and returns the read value. The value cannot be read and written across systems that have different endian values.

See also: ColReadDateTime (528), ColReadBoolean (527), ColReadString (529), ColReadInteger (528), ColWriteFloat (530), ColReadCurrency (528)

27.3.5 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 (530) and returns the read value. The value cannot be read and written across systems that have different endian values.

See also: ColReadDateTime (528), ColReadBoolean (527), ColReadString (529), ColWriteInteger (530), ColReadFloat (528), ColReadCurrency (528)

27.3.6 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 (530) and returns the read value. The value cannot be read and written across systems that have different endian values.

See also: ColReadDateTime (528), ColReadBoolean (527), ColWriteString (530), ColReadInteger (528), ColReadFloat (528), ColReadCurrency (528)

27.3.7 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 (527), ColWriteString (530), ColWriteInteger (530), ColWriteCurrency (529), ColWriteDateTime (529), ColWriteFloat (530)

27.3.8 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 (529), ColWriteString (530), ColWriteInteger (530), ColWriteDateTime (529), ColWriteFloat (530), ColReadCurrency (528)

27.3.9 ColWriteDateTime

Synopsis: Write a `TDateTime` value to stream

Declaration: `procedure ColWriteDateTime(S: TStream;AValue: TDateTime)`

Visibility: default

Description: `ColWriteDateTime` writes the `TDateTimeAValue` to the stream `S`.

See also: ColReadDateTime (528), ColWriteBoolean (529), ColWriteString (530), ColWriteInteger (530), ColWriteFloat (530), ColWriteCurrency (529)

27.3.10 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 (529), ColWriteBoolean (529), ColWriteString (530), ColWriteInteger (530), ColReadFloat (528), ColWriteCurrency (529)

27.3.11 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 (529), ColWriteString (530), ColReadInteger (528), ColWriteCurrency (529), ColWriteDateTime (529)

27.3.12 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 (529), ColReadString (529), ColWriteInteger (530), ColWriteCurrency (529), ColWriteDateTime (529), ColWriteFloat (530)

27.4 EStreamColl

27.4.1 Description

Exception raised when an error occurs when streaming the collection.

27.5 TStreamCollection

27.5.1 Description

`TStreamCollection` is a TCollection (**??**) descendent which implements 2 calls LoadFromStream (531) and SaveToStream (531) which load and save the contents of the collection to a stream.

The collection items must be descendents of the TStreamCollectionItem (532) class for the streaming to work correctly.

Note that the stream must be used to load collections of the same type.

27.5.2 Method overview

Page	Property	Description
531	LoadFromStream	Load the collection from a stream
531	SaveToStream	Load the collection from the stream.

27.5.3 Property overview

Page	Property	Access	Description
531	Streaming	r	Indicates whether the collection is currently being written to stream

27.5.4 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 (531). 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 (531)

27.5.5 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 (531). 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 (531)

27.5.6 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 (531), TStreamCollection.SaveToStream (531)

27.6 TStreamCollectionItem

27.6.1 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 (530) to save or load the item from the stream.

Chapter 28

Reference for unit 'streamex'

28.1 Used units

Table 28.1: Used units by unit 'streamex'

Name	Page
Classes	**??**

28.2 Overview

streamex implements some extensions to be used together with streams from the classes unit.

28.3 TBidirBinaryObjectReader

28.3.1 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.

28.3.2 Property overview

Page	Property	Access	Description
533	Position	rw	Position in the stream

28.3.3 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 (534) class.

See also: TDelphiReader (534)

28.4 TBidirBinaryObjectWriter

28.4.1 Description

`TBidirBinaryObjectReader` is a class descendent from TBinaryObjectWriter (**??**), which implements the necessary support for BiDi data.

28.4.2 Property overview

Page	Property	Access	Description
534	Position	rw	Position in the stream

28.4.3 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 (535) class.

See also: TDelphiWriter (535)

28.5 TDelphiReader

28.5.1 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 (533) driver class.

28.5.2 Method overview

Page	Property	Description
535	GetDriver	Return the driver class as a TBidirBinaryObjectReader (533) class
535	Read	Read data from stream
535	ReadStr	Overrides the standard `ReadStr` method

28.5.3 Property overview

Page	Property	Access	Description
535	Position	rw	Position in the stream

28.5.4 TDelphiReader.GetDriver

Synopsis: Return the driver class as a TBidirBinaryObjectReader (533) class

Declaration: `function GetDriver : TBidirBinaryObjectReader`

Visibility: public

Description: `GetDriver` simply returns the used driver and typecasts it as TBidirBinaryObjectReader (533) class.

See also: TBidirBinaryObjectReader (533)

28.5.5 TDelphiReader.ReadStr

Synopsis: Overrides the standard `ReadStr` method

Declaration: `function ReadStr : String`

Visibility: public

Description: `ReadStr` makes sure the TBidirBinaryObjectReader (533) methods are used, to store additional information about the stream position when reading the strings.

See also: TBidirBinaryObjectReader (533)

28.5.6 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 (533) class when reading.

See also: TBidirBinaryObjectReader (533), TDelphiReader.Position (535)

28.5.7 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 (535)

28.6 TDelphiWriter

28.6.1 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 (534) driver class.

28.6.2 Method overview

Page	Property	Description
536	FlushBuffer	Flushes the stream buffer
536	GetDriver	Return the driver class as a TBidirBinaryObjectWriter (534) class
536	Write	Write raw data to the stream
536	WriteStr	Write a string to the stream
537	WriteValue	Write value type

28.6.3 Property overview

Page	Property	Access	Description
537	Position	rw	Position in the stream

28.6.4 TDelphiWriter.GetDriver

Synopsis: Return the driver class as a TBidirBinaryObjectWriter (534) class

Declaration: `function GetDriver : TBidirBinaryObjectWriter`

Visibility: public

Description: `GetDriver` simply returns the used driver and typecasts it as TBidirBinaryObjectWriter (534) class.

See also: TBidirBinaryObjectWriter (534)

28.6.5 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.

28.6.6 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.

28.6.7 TDelphiWriter.WriteStr

Synopsis: Write a string to the stream

Declaration: `procedure WriteStr(const Value: String)`

Visibility: public

Description: `WriteStr` writes a string to the stream, forcing the use of the TBidirBinaryObjectWriter (534) class methods, which update the position of the stream.

See also: TBidirBinaryObjectWriter (534)

28.6.8 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 (534) methods, which update the position of the stream.

See also: TBidirBinaryObjectWriter (534)

28.6.9 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 (534) instance used when streaming.

See also: TBidirBinaryObjectWriter (534)

Chapter 29

Reference for unit 'StreamIO'

29.1 Used units

Table 29.1: Used units by unit 'StreamIO'

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

29.2 Overview

The StreamIO unit implements a call to reroute the input or output of a text file to a descendents of TStream (**??**).

This allows to use the standard pascal Read (**??**) and Write (**??**) functions (with all their possibilities), on streams.

29.3 Procedures and functions

29.3.1 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: #rtl.classes.TStream (**??**), GetStream (539)

29.3.2 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 (538).

Errors: An invalid class reference will be returned if the file was not associated with a stream.

See also: AssignStream (538), #rtl.classes.TStream (**??**)

Chapter 30

Reference for unit 'syncobjs'

30.1 Used units

Table 30.1: Used units by unit 'syncobjs'

Name	Page
sysutils	??

30.2 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 (541) class is a wrapper around low-level critical section routines (semaphores or mutexes). The TEventObject (543) class can be used to send messages between threads (also known as conditional variables in Posix threads).

30.3 Constants, types and variables

30.3.1 Constants

```
INFINITE = Cardinal(-1)
```

Constant denoting an infinite timeout.

30.3.2 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 (543) class.

```
TEventHandle = Pointer
```

`TEventHandle` is an opaque type and should not be used in user code.

```
TWaitResult = (wrSignaled,wrTimeout,wrAbandoned,wrError)
```

Table 30.2: Enumeration values for type TWaitResult

Value	Explanation
wrAbandoned	Wait operation was abandoned.
wrError	An error occurred during the wait operation.
wrSignaled	Event was signaled (triggered)
wrTimeout	Time-out period expired

`TWaitResult` is used to report the result of a wait operation.

30.4 TCriticalSection

30.4.1 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.

30.4.2 Method overview

Page	Property	Description
542	Acquire	Enter the critical section
543	Create	Create a new critical section.
543	Destroy	Destroy the criticalsection instance
542	Enter	Alias for `Acquire`
543	Leave	Alias for `Release`
542	Release	Leave the critical section
542	TryEnter	Try and obtain the critical section

30.4.3 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: TCriticalSection.Release (542)

30.4.4 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 immediatly.

See also: TCriticalSection.Acquire (542)

30.4.5 TCriticalSection.Enter

Synopsis: Alias for `Acquire`

Declaration: `procedure Enter`

Visibility: public

Description: `Enter` just calls Acquire (542).

See also: TCriticalSection.Leave (543), TCriticalSection.Acquire (542)

30.4.6 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 (542)

30.4.7 TCriticalSection.Leave

Synopsis: Alias for `Release`

Declaration: `procedure Leave`

Visibility: public

Description: `Leave` just calls Release (542)

See also: TCriticalSection.Release (542), TCriticalSection.Enter (542)

30.4.8 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: TCriticalSection.Destroy (543)

30.4.9 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: TCriticalSection.Create (543), TCriticalSection.Acquire (542)

30.5 TEventObject

30.5.1 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.

30.5.2 Method overview

Page	Property	Description
544	Create	Create a new event object
544	destroy	Clean up the event and release from memory
544	ResetEvent	Reset the event
545	SetEvent	Set the event
545	WaitFor	Wait for the event to be set.

30.5.3 Property overview

Page	Property	Access	Description
545	ManualReset	r	Should the event be reset manually

30.5.4 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: TEventObject.ManualReset (545), TEventObject.ResetEvent (544)

30.5.5 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: TEventObject.Create (544)

30.5.6 TEventObject.ResetEvent

Synopsis: Reset the event

Declaration:
```
procedure ResetEvent
```

Visibility: public

Description: `ResetEvent` turns off the event. Any WaitFor (545) operation will suspend the calling thread.

See also: TEventObject.SetEvent (545), TEventObject.WaitFor (545)

30.5.7 TEventObject.SetEvent

Synopsis: Set the event

Declaration: `procedure SetEvent`

Visibility: public

Description: `SetEvent` sets the event. If the ManualReset (545) is `True` any thread that was waiting for the event to be set (using WaitFor (545)) 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: TEventObject.WaitFor (545), TEventObject.ManualReset (545)

30.5.8 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 (545)) 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 (545)

30.5.9 TEventObject.ManualReset

Synopsis: Should the event be reset manually

Declaration: `Property ManualReset : Boolean`

Visibility: public

Access: Read

Description: Should the event be reset manually

30.6 THandleObject

30.6.1 Description

`THandleObject` is a parent class for synchronization classes that need to store an operating system handle. It introduces a property Handle (546) which can be used to store the operating system handle. The handle is in no way manipulated by `THandleObject`, only storage is provided.

30.6.2 Method overview

Page	Property	Description
546	destroy	Free the instance

30.6.3 Property overview

Page	Property	Access	Description
546	Handle	r	Handle for this object
546	LastError	r	Last operating system error

30.6.4 THandleObject.destroy

Synopsis: Free the instance

Declaration: `destructor destroy;  Override`

Visibility: public

Description: `Destroy` does nothing in the Free Pascal implementation of `THandleObject`.

30.6.5 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.

30.6.6 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 (546).

See also: THandleObject.Handle (546)

30.7 TSimpleEvent

30.7.1 Description

`TSimpleEvent` is a simple descendent of the TEventObject (543) class. It creates an event with no name, which must be reset manually, and which is initially not set.

30.7.2 Method overview

Page	Property	Description
547	Create	Creates a new `TSimpleEvent` instance

30.7.3 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 (544) with `Nil` for the security attributes, an empty name, `AManualReset` set to `True`, and `InitialState` to `False`.

See also: TEventObject.Create (544)

30.8 TSynchroObject

30.8.1 Description

`TSynchroObject` is an abstract synchronization resource object. It implements 2 virtual methods Acquire (547) which can be used to acquire the resource, and Release (547) to release the resource.

30.8.2 Method overview

Page	Property	Description
547	Acquire	Acquire synchronization resource
547	Release	Release previously acquired synchronization resource

30.8.3 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: TSynchroObject.Release (547)

30.8.4 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 (547) call.

See also: TSynchroObject.Acquire (547)

Chapter 31

Reference for unit 'URIParser'

31.1 Overview

The URIParser unit contains a basic type (TURI (548)) and some routines for the parsing (ParseURI (549)) and construction (EncodeURI (548)) 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..

31.2 Constants, types and variables

31.2.1 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 (549) call. It contains the contents of a URI, parsed out in it's various pieces.

31.3 Procedures and functions

31.3.1 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 (549)

31.3.2 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 (550)

31.3.3 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 (549), URIToFileName (550)

31.3.4 ParseURI

Synopsis: Parse a URI and split it into its constituent parts

Declaration:
```
function ParseURI(const URI: String) : TURI;  Overload
function ParseURI(const URI: String;const DefaultProtocol: String;
                  DefaultPort: Word) : 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 (548)

31.3.5 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`

31.3.6 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 (550), FilenameToURI (549)

Chapter 32

Reference for unit 'zstream'

32.1 Used units

Table 32.1: Used units by unit 'zstream'

Name	Page
Classes	**??**
gzio	551
zbase	551

32.2 Overview

The ZStream unit implements a TStream (**??**) descendent (TCompressionStream (552)) 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 (555)) and decompresses it with the inflate algorithm.

32.3 Constants, types and variables

32.3.1 Types

```
Tcompressionlevel = (clnone,clfastest,cldefault,clmax)
```

Compression level for the deflate algorithm

```
Tgzopenmode = (gzopenread,gzopenwrite)
```

Open mode for gzip file.

Table 32.2: Enumeration values for type Tcompressionlevel

Value	Explanation
cldefault	Use default compression
clfastest	Use fast (but less) compression.
clmax	Use maximum compression
clnone	Do not use compression, just copy data.

Table 32.3: Enumeration values for type Tgzopenmode

Value	Explanation
gzopenread	Open file for reading
gzopenwrite	Open file for writing

32.4 Ecompressionerror

32.4.1 Description

`ECompressionError` is the exception class used by the TCompressionStream (552) class.

32.5 Edecompressionerror

32.5.1 Description

`EDecompressionError` is the exception class used by the TDeCompressionStream (555) class.

32.6 Egzfileerror

32.6.1 Description

`Egzfileerror` is the exception class used to report errors by the Tgzfilestream (557) class.

32.7 Ezliberror

32.7.1 Description

Errors which occur in the zstream unit are signaled by raising an `EZLibError` exception descendent.

32.8 Tcompressionstream

32.8.1 Description

`TCompressionStream`

32.8.2 Method overview

Page	Property	Description
553	create	Create a new instance of the compression stream.
553	destroy	Flushe data to the output stream and destroys the compression stream.
554	flush	Flush remaining data to the target stream
554	get_compressionrate	Get the current compression rate
553	write	Write data to the stream

32.8.3 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: TCompressionStream.Destroy (553)

32.8.4 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: TCompressionStream.Create (553)

32.8.5 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 (552) exception is raised.

See also: TCompressionStream.Read (552), TCompressionStream.Seek (552)

32.8.6 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.

32.8.7 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.

32.9 Tcustomzlibstream

32.9.1 Description

`TCustomZlibStream` serves as the ancestor class for the TCompressionStream (552) and TDecompressionStream (555) classes.

It introduces support for a progess handler, and stores the input or output stream.

32.9.2 Method overview

Page	Property	Description
554	create	Create a new instance of `TCustomZlibStream`
555	destroy	Clear up instance

32.9.3 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 (552), TDecompressionStream (555)

32.9.4 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 (554)

32.10 Tdecompressionstream

32.10.1 Description

`TDecompressionStream` performs the inverse operation of TCompressionStream (552). 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 (552) compression stream.

32.10.2 Method overview

Page	Property	Description
555	create	Creates a new instance of the `TDecompressionStream` stream
555	destroy	Destroys the `TDecompressionStream` instance
557	get_compressionrate	Get the current compression rate
556	read	Read data from the compressed stream
556	seek	Move stream position to a certain location in the stream.

32.10.3 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: TDecompressionStream.Destroy (555)

32.10.4 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: TDecompressionStream.Create (555)

32.10.5 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 (552) exception is raised.

See also: TCompressionStream.Write (553)

32.10.6 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 (552) exception is raised if the stream does not allow the requested seek operation.

See also: TDecompressionStream.Read (556)

32.10.7 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.

32.11 TGZFileStream

32.11.1 Description

`TGZFileStream` can be used to read data from a gzip file, or to write data to a gzip file.

32.11.2 Method overview

Page	Property	Description
557	create	Create a new instance of `TGZFileStream`
558	destroy	Removes `TGZFileStream` instance
557	read	Read data from the compressed file
558	seek	Set the position in the compressed stream.
558	write	Write data to be compressed

32.11.3 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 (557) 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 (558) method will be stored in the file in compressed (deflated) form.

Errors: If the file is not found, an EZlibError (552) exception is raised.

See also: TGZFileStream.Destroy (558), TGZOpenMode (551)

32.11.4 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: TGZFileStream.Create (557), TGZFileStream.Write (558), TGZFileStream.Seek (558)

32.11.5 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 (552) exception is raised.

See also: TGZFileStream.Create (557), TGZFileStream.Read (557), TGZFileStream.Seek (558)

32.11.6 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 (556) for a list of possibilities.

Errors: In case an impossible combination is asked, an EZlibError (552) exception is raised.

See also: TDecompressionStream.Seek (556)

32.11.7 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: TGZFileStream.Create (557)